
# 《GIS 设计与开发》课程设计报告

（2023—2024 年学年第二学期）

| | |
|---|---|
| 题　　目 | 遥感分类系统设计与开发 |
| 专　　业 | 空间信息与数字技术 |
| 班　　级 | 空间信息与数字技术 A |
| 姓　　名 | 2243206000333 周泽同 |
| | 2243206000307 李普通 |
| | 2243206000324 罗祥澄 |

# 《GIS 设计与开发》课程设计

# 成 绩 评 定

| 学号 | 姓名 | 成绩 | 备注 |
|---|---|---|---|
| 2243206000333 | 周 泽 同 | | 组长，组织工作，答辩汇报，系统界面框架设计及基本功能的实现，核心功能影像裁剪和随机森林监督分类算法的实现，开发文档和汇报 PPT 的撰写。 |
| 2243206000307 | 李 普 通 | | 核心功能 NDVI 及其重分类和非监督分类算法的实现，开发文档的撰写。 |
| 2243206000324 | 罗 祥 澄 | | 开发文档及汇报 PPT 的撰写，数据收集。 |

指导教师： 朱彦辉

# 目录

# 1 系统背景分析

## 1.1 研究背景

在信息化与数字化的浪潮中，Python 作为一种简洁、高效且易于学习的编程语言，在数据处理、科学计算及人工智能等领域展现出了强大的潜力和广泛的应用前景。特别是在地理信息系统（GIS）与遥感技术快速发展的今天，Python 凭借其丰富的库和框架，如 GDAL、Rasterio、Scikit-learn 等，为遥感数据的处理与分析提供了强有力的支持。

随着遥感数据源的不断增加和多样化，如何高效地处理、分析和利用这些数据成为了科研工作者和实际应用者面临的重大挑战。传统的 GIS 软件虽然功能强大，但在处理复杂、大规模遥感数据时，往往存在运行效率低、定制化难度大等问题。因此，利用 Python 开发一款灵活、高效、可扩展的遥感数据处理与分析软件显得尤为重要。

本项目旨在运用 Python+QGIS 开发一款综合性的遥感数据处理与分析软件。该软件不仅集成了 QGIS 软件的基础功能，如地图浏览、图层管理等，还深入挖掘了 Python 在遥感数据处理领域的优势，实现了 NDVI 计算、非监督分类、重分类以及随机森林分类等高级分析功能。通过 Python 的脚本化特性，用户可以轻松自定义处理流程，实现数据处理的自动化和批量化。

在 NDVI 计算方面，Python 的 NumPy 和 SciPy 等库提供了高效的数值计算能力，使得 NDVI 的计算更加迅速、准确。同时，通过结合 Rasterio 等库，可以轻松读取和写入多种格式的遥感图像文件，实现数据的无缝对接。

在非监督分类和随机森林分类方面[1]，Python 的 Scikit-learn 库提供了丰富的机器学习算法，包括聚类算法和决策树、随机森林等分类算法。这些算法的应用使得图像分类过程更加智能化、精准化。用户可以根据实际需求选择合适的分类方法，并通过调整参数来优化分类效果。

此外，Python 的开源特性也为该软件的持续发展和完善提供了保障。通过不断吸收社区中的优秀算法和工具，该软件将能够保持其领先地位，并满足用户日益增长的需求。

## 1.2 需求分析

### 1.2.1 项目背景与目标

随着遥感技术的快速发展和广泛应用，遥感数据的处理与分析已成为科研、环境监测、城市规划、农业管理等多个领域的重要工作。然而，现有的遥感数据处理软件往往存在功能单一、操作复杂、定制化程度低等问题，难以满足用户多样化的需求。因此，本项目旨在开发一款基于 Python+QGIS 的定制化遥感数据处理与分析软件，以提高遥感数据处理的效率、准确性和灵

活性。

### 1.2.2 用户需求分析

1）基础 GIS 功能需求：其中包括打开工程文档，导出地图，地图拖动、放大和缩小，矢量编辑等基本功能。

2）高级分析功能需求：NDVI 计算，用户需要能够自动计算归一化植被指数（NDVI），以评估植被覆盖和生长状况；非监督分类[2]，提供自动化的非监督分类算法，如 K-means 聚类等，以便用户能够快速对遥感图像进行分类；重分类，用户需要能够对分类结果进行重分类，以满足特定的分析需求；随机森林分类，提供随机森林分类算法，以提高分类的准确性和稳定性。

3）性能与易用性需求：软件应具备良好的性能，能够处理大规模遥感数据而不卡顿；用户界面应简洁明了，易于操作，降低学习成本；提供详细的帮助文档和教程，以便用户能够快速上手。

4）扩展性与兼容性需求：软件应具备良好的扩展性，能够支持未来新算法的添加和现有功能的优化；确保软件与主流操作系统和硬件平台兼容，以满足不同用户的使用需求。

5）系统需求：硬件需求，推荐配置为高性能计算机，具备足够的内存和存储空间以处理大规模遥感数据；软件需求，基于 Python 开发，但无需安装 Python 环境及相关的库和框架（如 GDAL、Rasterio、Scikit-learn 等），无需 QGIS 的相关开发包，开发者使用 pyinstaller 将环境与应用源代码打包成可执行程序，使用户可以在不具备相关开发环境的情况下正常使用系统；网络需求，软件应支持离线使用，但也可考虑集成网络功能以便用户能够下载和上传遥感数据。

6）非功能性需求：安全性，确保软件在处理敏感遥感数据时能够保护用户隐私和数据安全；可靠性，软件应具备良好的稳定性和可靠性，减少崩溃和错误的发生；可维护性，软件代码应结构清晰、易于理解和维护。

# 2 系统技术分析

## 2.1 系统功能设计

1）本系统的功能模块图如下：

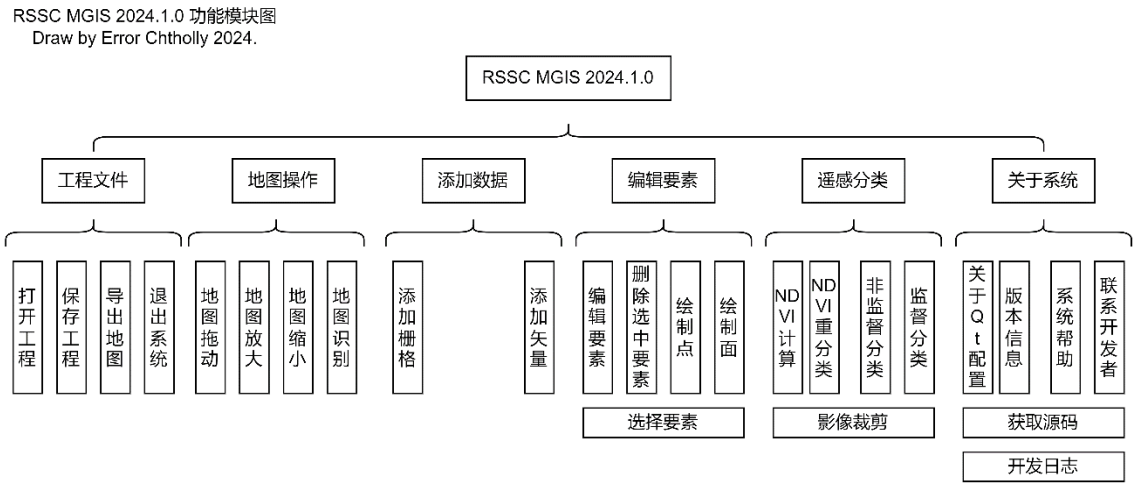



**图 1　功能模块图**

**Fig. 1　Functional module diagram**

2）QGIS 基础功能集成。

3）NDVI 计算[3]。

数据源选择：允许用户选择包含红光波段和近红外波段的遥感影像数据源。

参数设置：提供界面让用户设置导入的数据和输出的目录。

计算执行：执行 NDVI 计算，生成 NDVI 影像。

结果展示：将 NDVI 计算结果以图层形式展示在地图上，支持颜色映射（如绿色表示高植被覆盖，红色表示低植被覆盖）。

4）非监督分类。

数据源准备：要求用户选择需要进行分类的遥感影像或数据集。

算法选择：提供多种非监督分类算法供用户选择（如 K-means、层次聚类等）。

参数配置：允许用户设置输入文件和输出路径。

分类执行：执行非监督分类算法，生成分类结果。

结果评估：提供工具或界面让用户评估分类结果的准确性（如混淆矩阵、分类精度等）。

5）NDVI 重分类。

数据源选择：允许用户选择需要进行重分类的栅格数据（如 NDVI 影像、分类结果等）。

规则定义：提供界面让用户选择输入文件和输出路径。

重分类执行：根据定义的规则执行重分类操作，生成新的栅格数据。

结果展示：将重分类结果以图层形式展示在地图上。

6）随机森林监督分类。

数据源准备：要求用户具备要分类的遥感图像和人工选择好的分类示例文件。

算法选择：提供 scikit-learn 的监督分类深度学习算法库。

分类执行：执行监督分类算法，生成分类结果。

结果展示：将监督分类结果以图层形式展示在地图上。

## 2.2 数据库设计

本系统的相关算法和操作均基于算法库和文件方式进行操作，无需进行数据库的设计。

# 3 系统实现过程

## 3.1 环境配置

1） PyQGIS 环境配置：本系统使用 QGIS 3.34.8 LTR 版本的 Python 3.12 二次开发库进行基本功能的开发。

2）PyCharm+PyQt 环境配置本系统使用 PyCharm 2024 搭配 QGIS 提供的 PyQt 5 使用，进行用户 Gui 界面的开发和设计。

3）深度学习环境配置：本系统使用 rasterio 和 scikit-learn 深度学习算法库进行遥感分类识别高级分析功能的开发。

## 3.2 系统实现过程

### 3.2.1 基础功能的实现

通过使用 QGIS 提供的二次开发库进行了基础功能的开发，部分代码如下：

```python
class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        vl = QVBoxLayout(self.dockWidgetContents)
        self.layerTreeView = QgsLayerTreeView(self)
        vl.addWidget(self.layerTreeView)
        # 3 初始化地图画布
        self.mapCanvas = QgsMapCanvas(self)
        hl = QHBoxLayout(self.frame)
        hl.setContentsMargins( *__args: 0,0,0,0) #设置周围间距
        hl.addWidget(self.mapCanvas)
        # 4 设置图层树风格
        self.model = QgsLayerTreeModel(PROJECT.layerTreeRoot(),self)
        self.model.setFlag(QgsLayerTreeModel.AllowNodeRename) #允许图层节点重命名
        self.model.setFlag(QgsLayerTreeModel.AllowNodeReorder) #允许图层拖拽排序
        self.model.setFlag(QgsLayerTreeModel.AllowNodeChangeVisibility) #允许改变图层节点可视性
        self.model.setFlag(QgsLayerTreeModel.ShowLegendAsTree) #展示图例
        self.model.setAutoCollapseLegendNodes(10) #当节点数大于等于10时自动折叠
        self.layerTreeView.setModel(self.model)
        # 4 建立图层树与地图画布的桥接
        self.layerTreeBridge = QgsLayerTreeMapCanvasBridge(PROJECT.layerTreeRoot(),self.mapCanvas,self)
        # 5 初始加载影像
        self.firstAdd = True
        # 6 允许拖拽文件
        self.setAcceptDrops(True)
        # 7 图层树右键菜单创建
        self.rightMenuProv = menuProvider(self)
        self.layerTreeView.setMenuProvider(self.rightMenuProv)
        # 8.0 提前给予基本CRS
        self.mapCanvas.setDestinationCrs(QgsCoordinateReferenceSystem("EPSG:4490"))
        # 8 状态栏控件
        self.statusBar = QStatusBar()
        self.statusBar.setStyleSheet('color: rgb(255, 255, 255); border: rgb(255, 255, 255); background-color:rgb(170, 170, 255); font: 9pt "幼圆" ;')
        self.statusXY = QLabel('{:<40}'.format(''))  # x y 坐标状态
        self.statusBar.addWidget(self.statusXY, QWidget: 1)
        self.statusScaleLabel = QLabel('比例尺')
        self.statusScaleComboBox = QComboBox(self)
        self.statusScaleComboBox.setFixedWidth(120)
        self.statusScaleComboBox.addItems(
            ["1:500", "1:1000", "1:2500", "1:5000", "1:10000", "1:25000", "1:100000", "1:500000", "1:1000000"])
        self.statusScaleComboBox.setEditable(True)
```

图层右键菜单：

```python
class menuProvider(QgsLayerTreeViewMenuProvider):
    def createContextMenu(self) -> QtWidgets.QMenu:
        if len(self.layerTreeView.selectedLayers()) > 1:
            # 添加组
            self.actionGroupSelected = self.actions.actionGroupSelected()
            menu.addAction(self.actionGroupSelected)

            self.actionZoomToLayer = self.actions.actionZoomToLayer(self.mapCanvas, menu)
            menu.addAction(self.actionZoomToLayer)

            self.actionMoveToTop = self.actions.actionMoveToTop(menu)
            menu.addAction(self.actionMoveToTop)

            actionDeleteSelectedLayers = QAction( *__args: '删除选中图层(D)', menu)
            actionDeleteSelectedLayers.triggered.connect(self.deleteSelectedLayer)
            menu.addAction(actionDeleteSelectedLayers)

            return menu

        node: QgsLayerTreeNode = self.layerTreeView.currentNode()
        if node:
            if QgsLayerTree.isGroup(node):
                group: QgsLayerTreeGroup = self.layerTreeView.currentGroupNode()
                self.actionRenameGroup = self.actions.actionRenameGroupOrLayer(menu)
                menu.addAction(self.actionRenameGroup)

                self.actionMoveToTop = self.actions.actionMoveToTop(menu)
                menu.addAction(self.actionMoveToTop)

                actionDeleteGroup = QAction( *__args: '删除组(D)', menu)
                actionDeleteGroup.triggered.connect(lambda: self.deleteGroup(group))
                menu.addAction(actionDeleteGroup)
            elif QgsLayerTree.isLayer(node):
                layer: QgsMapLayer = self.layerTreeView.currentLayer()

                if layer.type() == QgsMapLayerType.VectorLayer:
                    actionOpenAttributeDialog = QAction( *__args: '打开属性表', menu)
                    actionOpenAttributeDialog.triggered.connect(lambda: self.openAttributeDialog(layer))
                    menu.addAction(actionOpenAttributeDialog)

                self.actionZoomToLayer = self.actions.actionZoomToLayer(self.mapCanvas, menu)
```

3.2.2 NDVI 算法实现

1）原理：NDVI 是通过遥感图像技术来评估植被健康的指标。它基于植被在可见光和红外光波段的反射特性差异来计算。具体来说，植被吸收红光（RED）而反射近红外光（NIR）。因此，在一个健康的植被区域中，反射红外光的相对强度会比红光更强。NDVI 的计算公式为：

$$NDVI=(NIR-R)/(NIR+R)$$

注：NIR 是近红外波段，R 是红外波段

这个公式反映了近红外波段与红光波段的反射率差异，从而可以评估植被的生长状况和健康程度 。

2）计算步骤：

数据准备：首先需要准备红光波段和近红外波段的遥感影像数据。这些数据通常以图像文件（如 GeoTIFF）的形式存在。

读取数据：使用 Python 中的库（如 rasterio、gdal 等）读取红光波段和近红外波段的遥感影像数据。这些库能够处理多种格式的遥感图像，并方便地读取图像的波段数据。

数据处理：将读取的波段数据转换为 NumPy 数组，以便进行数值计算。在转换过程中，可能需要进行一些预处理操作，如去噪、裁剪等。

计算 NDVI：根据 NDVI 的计算公式，对红光波段和近红外波段的数据进行计算，得到 NDVI 值。这个计算过程通常涉及到数组的逐元素运算。

结果可视化：将计算得到的 NDVI 值映射到图像上，并使用适当的颜色映射表（colormap）进行可视化。这有助于直观地展示植被的分布和生长状况。

结果分析：对可视化后的 NDVI 图像进行分析，评估植被的覆盖情况、生长状态以及可能的生态环境变化。

3）Python 算法实现：在 Python 中，可以使用 rasterio 和 numpy 库来实现 NDVI 的计算。部分相关代码如下：

```
#计算NDVI值并显示
    1 个用法
    def updateNdviResult(self):
        redBand = rio.open(self.NDVI.redBandPath)
        nirBand = rio.open(self.NDVI.nirBandpath)
        red = redBand.read(1).astype("float64")
        nir = nirBand.read(1).astype("float64")
        ndvi = np.where(
            (nir+red)==0.,0,
            (nir-red)/(nir+red))
        kwargs =redBand.meta
        kwargs.update(
            dtype=rio.float32,
            count =1,
            compress='lzw'
        )
        output =self.NDVI.outputPath+"/ndvi_result.tif"
        with rio.open(output, mode: 'w',**kwargs)as dst:
            dst.write_band(1,ndvi.astype(rio.float32))
        if output:
            rasterLayer = readRasterFile(output)
            if rasterLayer.isValid():
                if self.firstAdd:
                    addMapLayer(rasterLayer, self.mapCanvas, firstAddLayer: True)
                    self.firstAdd = False
                else:
                    addMapLayer(rasterLayer, self.mapCanvas)
```

3.2.3 NDVI 重分类

1）确定重分类规则：首先，需要确定如何将原始的 NDVI 值映射到新的类别或值范围。这通常基于 NDVI 值的生态学意义或分析需求。

2）应用重分类规则：使用选定的规则对 NDVI 图像中的每个像素值进行重分类。

3）生成重分类后的图像：将重分类后的像素值组合成新的图像。

4）Python 实现：在 Python 中，可以使用 rasterio 和 numpy 库来读取和处理 NDVI 图像，并使用 numpy 的数组操作功能来应用重分类规则。部分相关代码如下：

```
def reclass(self):
    def classify_ndvi(ndvi):
        classes =np.empty_like(ndvi,dtype=int)
        classes[ndvi<0.1]=1
        classes[(ndvi>=0.1)&(ndvi<0.4)]=2
        classes[(ndvi>=0.4)&(ndvi<0.7)]=3
        classes[ndvi>=0.7]=4
        return classes
    with rio.open(self.NDVI_RECLASS.NdviPath) as src:
        ndvi_array=src.read(1)
        profile =src.profile
        classified_array=classify_ndvi(ndvi_array)
        profile.update(
            dtype=rio.uint8,
            count=1
        )
        metadata={
            'TIFFTAG_DOCUMENTNAME': 'Classified NDVI',
            'CLASSIFICATION': '1=LOW Vegetation,2=Sparse Vegetation,3=Moderate Vegetation,4=Hige Vegetation'
        }
        with rio.open(self.NDVI_RECLASS.OutputPath+"/classified_ndvi.tif",mode='w',**profile,metadata=metadata) as dst:
            dst.write(classified_array.astype(rio.uint8),1)
            layer_path = self.NDVI_RECLASS.OutputPath+"/classified_ndvi.tif"
            layer = QgsRasterLayer(layer_path, 'Classified Raster')
            if not layer.isValid():
                print("Layer failed to load!")
            else:
                colorRampShader = QgsColorRampShader()
                colorRampShader.setColorRampItemList([
                    QgsColorRampShader.ColorRampItem(0, QColor( *args 128, 128, 0)),  # 红色
                    QgsColorRampShader.ColorRampItem(1, QColor( *args 153, 255, 0)),  # 绿色
                    QgsColorRampShader.ColorRampItem(2, QColor( *args 0, 184, 255)),  # 蓝色

                ])
                rasterShader = QgsRasterShader()
                rasterShader.setRasterShaderFunction(colorRampShader)
                renderer = QgsSingleBandPseudoColorRenderer(
                    layer.dataProvider(),
                    1,
                    rasterShader
                )
                layer.setRenderer(renderer)
                QgsProject.instance().addMapLayer(layer)
            output=self.NDVI_RECLASS.OutputPath+"/classified_ndvi.tif"
        if output:
            rasterLayer = readRasterFile(output)
            if rasterLayer.isValid():
                if self.firstAdd:
                    addMapLayer(rasterLayer, self.mapCanvas, firstAddLayer: True)
                    self.firstAdd = False
                else:
                    addMapLayer(rasterLayer, self.mapCanvas)
```

3.2.4 非监督分类

1）定义与原理：非监督分类是指人们事先对分类过程不施加任何的先验知识，而仅凭数据，即自然聚类的特性，进行“盲目”的分类。其分类的结果只是对不同类别达到了区分，但并不能确定类别的属性，类别的属性是通过分类结束后目视判读或实地调查确定的。

2）常用算法：非监督分类的算法多种多样，常见的有以下几种。

ISODATA 分类：全称“迭代自组织数据分析技术”（Iterative Self-Organizing Data Analysis Technique）。使用最小光谱距离方程产生聚类，以随机的类中心作为初始类别的“种子”，依据某个判别规则进行自动迭代聚类的过程。在两次迭代的之间对上一次迭代的聚类结果进行统计分析，根据统计参数对已有类别进行取消、分裂、合并处理，并继续进行下一次迭代，直至超过最大迭代次数或者满足分类参数（阈值），完成分类过程。

K-均值（K-Means）分类：也称 c-均值算法，是一种迭代求解的聚类分析算法。其基本思想是，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。具体步骤包括计算特征空间上均匀分布的最初类均值，用最短距离技术重复地把像元聚集到最近的类里，每次迭代重新计算均值，并用这一新的均值对像元进行再分类，直到每一类的像元数变化少于选择的像元变化阈值或已经达到了迭代的最多次数。

其他算法：还包括回归分析、趋势分析、等混合距离法、主成分分析和图形识别等。在实际应用中，还可能使用到模糊聚类法、系统聚类、链状方法等。

3）分类步骤：遥感影像的非监督分类一般包括以下步骤。

影像分析：大体上判断主要地物的类别数量。

分类器选择：根据需求选择合适的非监督分类算法，如 ISODATA 或 K-Means。

设置参数：在分类器中设置类别数目、迭代次数等参数。

执行分类：根据设置的参数执行非监督分类。

分类结果分析：对分类结果进行分析，确认各类别的实际属性。

类别定义与合并：对分类结果进行目视判读或实地调查，定义各类别的属性，并进行必要的类别合并。

4）Python 实现：部分相关代码如下。

```
    def imageunsupervise(self):
        rasterFilePath=self.UNSUPERVISE.reclassPath
        with rio.open(rasterFilePath) as src:
            band = src.read(1)
        band_normalized = (band - band.min()) / (band.max() - band.min())
        k = 5
        kmeans = KMeans(n_clusters=k, random_state=0).fit(band_normalized.reshape(-1, 1))
        labels = kmeans.labels_
        labels_2d = labels.reshape(band.shape)
        with rio.open(self.UNSUPERVISE.outputPath+"/UnsupervisedResult.tif", mode='w', driver='GTiff',
                      height=src.height, width=src.width, count=1, dtype=np.uint8,
                      crs=src.crs, transform=src.transform) as dst:
            dst.write(labels_2d.astype(np.uint8), 1)
            layer_path= self.UNSUPERVISE.outputPath+"/UnsupervisedResult.tif"
            layer = QgsRasterLayer(layer_path, 'Classified Raster')
            if not layer.isValid():
                print("Layer failed to load!")
            else:
                colorRampShader = QgsColorRampShader()
                colorRampShader.setColorRampItemList([
                    QgsColorRampShader.ColorRampItem(0, QColor( *args 128, 128, 0)),  # 红色
                    QgsColorRampShader.ColorRampItem(1, QColor( *args 198, 60, 0)),  # 黄色
                    QgsColorRampShader.ColorRampItem(2, QColor( *args 135, 255, 0)),  # 绿色
                    QgsColorRampShader.ColorRampItem(3, QColor( *args 0, 184, 255)),  # 蓝色
                    QgsColorRampShader.ColorRampItem(4, QColor( *args 255, 99, 123))  # 品红色
                ])
                rasterShader = QgsRasterShader()
                rasterShader.setRasterShaderFunction(colorRampShader)
                renderer = QgsSingleBandPseudoColorRenderer(
                    layer.dataProvider(),
                    1,
                    rasterShader
                )
                layer.setRenderer(renderer)
                QgsProject.instance().addMapLayer(layer)
```

3.2.5 随机森林监督分类

1）基本思想：随机森林就是通过集成学习的思想将多棵树集成的一种算法，它的基本单元是决策树，而它的本质属于机器学习的一大分支——集成学习（Ensemble Learning）方法。随机森林的名称中有两个关键词，一个是“随机”，一个就是“森林”。“森林”我们很好理解，一棵叫做树，那么成百上千棵就可以叫做森林了，这样的比喻还是很贴切的，其实这也是随机森林的主要思想－集成思想的体现。

从直观角度来解释，每棵决策树都是一个分类器（假设现在针对的是分类问题），那么对于一个输入样本，N 棵树会有 N 个分类结果。而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出，这就是一种最简单的 Bagging 思想。

2）优缺点：每棵树都选择部分样本及部分特征，一定程度避免过拟合；每棵树随机选择样本并随机选择特征，使得具有很好的抗噪能力，性能稳定；能处理很高维度的数据，并且不用做特征选择（不需要降维处理）；适合并行计算；实现比较简单。但参数较复杂，模型训练和预测都比较慢。

3）Python 算法实现：使用 rasterio 和 scikit-learn 深度学习算法库实现，部分相关代码如下。

```
class MainWindow(QMainWindow, Ui_MainWindow):
    1 个用法
    def imageRF(self):
        #dataset_list = [rio.open(f'./data/band{i}.tif') for i in range(2, 8)]
        dataset_list = [rio.open(self.randomForestDialog.rasterPath+f'./band{i}.tif') for i in range(2, 8)]
        calgary_trainingpointer_gpd = gpd.read_file(self.randomForestDialog.rasterPath)
        all_read_vector = np.concatenate([dataset_list[i].read() for i in range(len(dataset_list))], axis=0)

        def location2value(x, y):
            row, col = dataset_list[0].index(x, y)
            res = all_read_vector[:, row, col]
            return pd.Series(res)

        trainX = calgary_trainingpointer_gpd.to_crs(dataset_list[0].crs.to_string()).pipe(
            lambda x: x.assign(**{'lon': x.geometry.x, 'lat': x.geometry.y})).pipe(
            lambda x: x.apply(lambda x: location2value(x['lon'], x['lat']), axis=1))
        trainY = calgary_trainingpointer_gpd['class']
        X_train, X_test, y_train, y_test = train_test_split( *arrays: trainX, trainY, train_size=0.8, random_state=42)
        rf_fit = RandomForestClassifier()  # SVC()
        rf_fit.fit(X_train, y_train)
        predict_all_x = np.hstack([dataset_list[i].read().reshape(-1, 1) for i in range(len(dataset_list))])
        predict_all_result = rf_fit.predict(predict_all_x)
        class_list = np.unique(predict_all_result).tolist()
        class_dict = {value: index + 1 for index, value in enumerate(class_list)}
        result = pd.DataFrame({'class': predict_all_result})['class'].map(class_dict).values.reshape(
            dataset_list[0].read().shape[1:])
        result_mask = result.copy().astype(np.float64)
        prof = dataset_list[0].profile

        # 更新 profile 中的数据类型以匹配分类结果
        prof.update(
            dtype=rio.uint8,  # 假设我们使用 uint8 存储分类结果
            count=1  # 分类结果是一个波段
        )
        metadata = {
            'TIFFTAG_DOCUMENTNAME': 'Classified NDVI',
            'CLASSIFICATION': '1=water, 2=forest, 3=land, 4=building'
        }
        # 写入新的栅格文件
        with rio.open(self.randomForestDialog.rasterPath+'/rf_result.tif',  mode: 'w', **prof, metadata=metadata) as dst:
            dst.write(result_mask.astype(rio.uint8), 1)
```

### 3.2.6 遥感图像裁剪

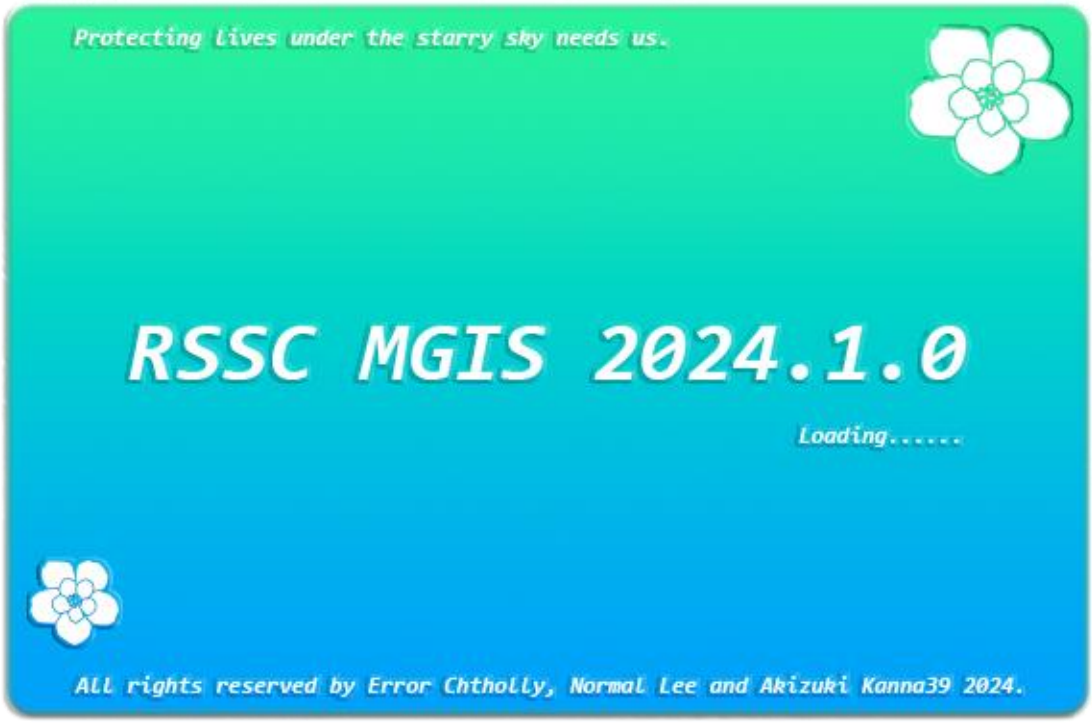
部分相关代码如下：

```
1 个用法
def updateCutResult(self):
    layer=QgsRasterLayer(self.cutDialog.rasterPath,os.path.basename(self.cutDialog.rasterPath))
    layervector=QgsVectorLayer(self.cutDialog.vectorPath,os.path.basename(self.cutDialog.vectorPath))
    addMapLayer(layer,self.mapCanvas)
    addMapLayer(layervector,self.mapCanvas)
    path=Path(self.cutDialog.rasterPath)
    fname=path.stem
    outpath=self.cutDialog.outputPath+"/"+fname+"_cut.tif"
    OutTile=gdal.Warp(outpath,self.cutDialog.rasterPath,cutlineDSName=self.cutDialog.vectorPath,cropToCutline=True,dstNodata=0)
    OutTile=None
    if outpath:
        rasterLayer=readRasterFile(outpath)
        if rasterLayer.isValid():
            if self.firstAdd:
                addMapLayer(rasterLayer,self.mapCanvas, firstAddLayer: True)
                self.firstAdd=False
            else:
                addMapLayer(rasterLayer,self.mapCanvas)
```

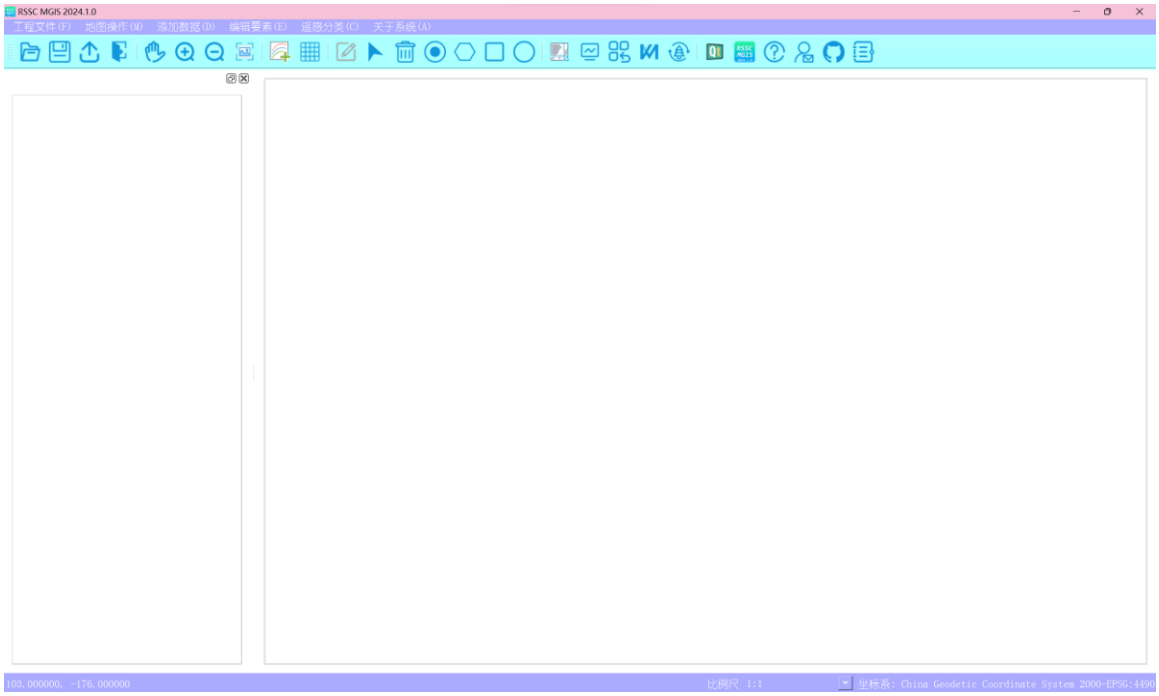
# 4 界面设计

## 4.1 系统图标设计（系统图标由开发者独立设计）

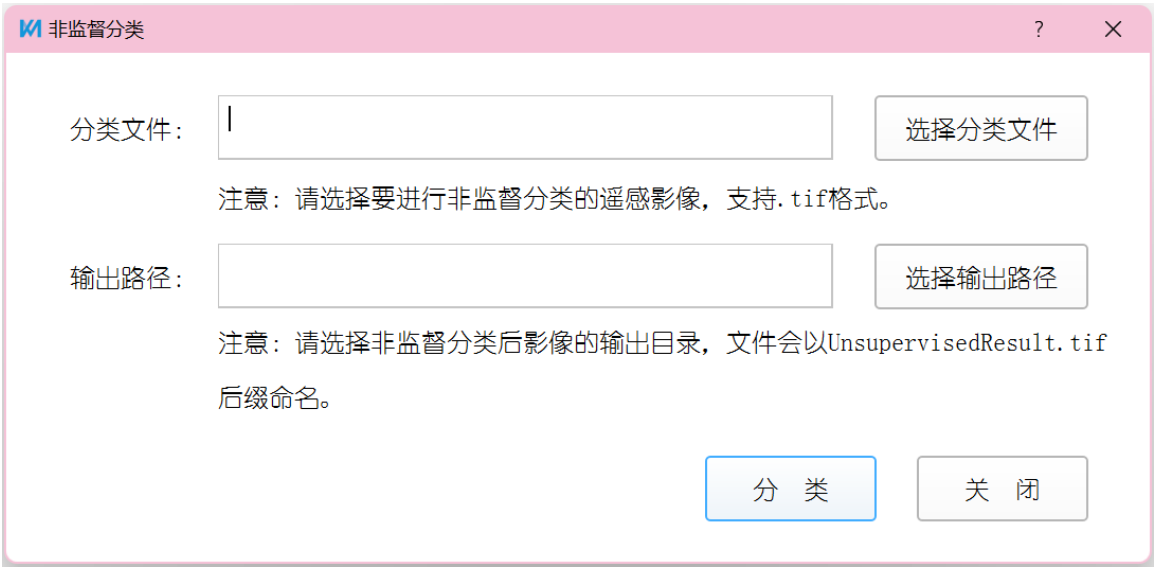

## 4.2 启动画面设计（启动画面由开发者独立设计）



## 4.3 主界面设计




## 4.4 部分子系统界面设计





## 5 数据来源

系统中所使用的遥感图像数据均来自地理空间数据云（地理空间数据云 (gscloud.cn)）。矢量数据均来自 www.poi86.com 数据网站。

## 6 系统总结

1）本系统实现了 QGIS 的部分基础功能，以及实现了定制化的遥感图像识别分类核心高级

分析功能，包括 NDVI 提取、NDVI 重分类、非监督分类、遥感影像裁剪、随机森林监督分类算法。

2）本系统界面整体采用简洁的设计风格，使用户使用时可以享受简单愉快的视觉体验。启动界面和应用图标采用蓝青渐变色风格和白色字体，简约的同时体现出一种轻量化的架构设计。主界面采用蓝紫配色的轻量化设计，子界面采用纯白配色简约风格。

3）本系统执行相关分类算法的处理时间整体较为迅速，但需要根据数据处理的数据量来进行动态分析和调整。

# 7 独立应用打包测试

为了使不具备相关 Python 和 QGIS 环境和库的客户机上能够运行该应用程序，本系统使用 pyinstaller 进行可执行程序的打包，并使用 Inno Setup 打包工具制作成安装包发布给电脑上无相关环境的客户进行使用测试，结果表明，该应用程序打包后可以在不具备相关环境的客户机上独立运行和使用。

# 8 参考文献

[1] 李红霞,石云,沈爱红,等.基于随机森林算法的贺兰山东麓洪积扇微地形分类研究[J/OL].第四纪研究,1-13[2024-07-05].http://kns.cnki.net/kcms/detail/11.2708.P.20240701.1519.002.html.

[2] 常婉秋,姚宇,席晓杰,等.综合迁移学习和非监督分类的制种玉米遥感识别方法研究[J/OL].农业机械学报,1-20[2024-07-05].http://kns.cnki.net/kcms/detail/11.1964.s.20240624.1526.019.html.

[3] 马靖,孙桂丽,禹明柱,等.基于植被指数的昌吉州荒漠化时空变化监测与评价[J].防护林科技,2024,(04):36-43+81.DOI:10.13601/j.issn.1005-5215.2024.04.010.

[4] PyQGIS 环境配置参考网址：https://www.zhihu.com/column/c_1641448508350812161

[5] 随机森林监督分类参考网址：https://blog.csdn.net/u014444411/article/details/108357109

[6] PyQGIS 部分基本功能实现参考网址：https://github.com/luolingchun/PyQGIS