2021.12

# 机器学习课程设计

## 植物幼苗分类-实验报告

黄元通 20195063

邱聖邦 20195263

# 目录

# 实验 4 植物幼苗分类

## 一、基本信息

### 1、成员及分工

队长：黄元通 学号：20195063 工作量：50%

组员：邱聖邦 学号：20195263 工作量：50%

分工：

黄元通：SIFT+词袋模型特征提取及降维；图片预处理；数据加载；K 折交叉验证；test 集预测标签生成；程序整合及修订。

邱聖邦：HOG 特征提取及降维；LBP 特征提取及降维；分类器调整与集成；特征融合。

### 2、概要

本次机器学习实验主要围绕“图片预处理”、“多种特征提取及特征融合”、“分类器选取与集成”三大部分展开。

通过控制变量法的实验，本文实验并选取了能最有效提高模型准确率的图片预处理方式：先均衡化，再去除背景。相对于均衡化、去背景、锐化操作的其他顺序组合，从提高模型正确率角度，其体现出了最好的图片预处理效果。

通过实验测试了单独使用“SIFT 特征+词袋模型”、“HOG 特征+PCA 降维”、“LBP 特征+直方图统计降维+PCA 降维”三种方式进行特征提取时的模型准确率，并测试证明了将三种方式得到的特征通过横向连接以实现特征融合，对模型准确率的有效提高作用。

通过在验证集上的正确率、F1-score 等性能衡量指标，测试使用了共 6 种分类器，最终从其中选取了 SVC 支持向量机、随机森林、ExtraTree 分类器、XGBoost 这四种分类器进行集成。并在之后根据各性能衡量指标，对集成中的各类分类器的参数进行了调整。

最终模型在验证集上的精确率可达到 0.9068，F1-score 可达到 0.86075；在测试集上，Kaggle 的 F1-score 打分可达 0.75755。

### 3、程序及各模块介绍

程序运行方法：

将原始数据集命名为“data”，并和所有程序文件放于同一路径下，运行“main.py”模块即可。

程序总体运行逻辑如下：

1. 先从 data 文件夹读取图片将图片进行预处理后保存在 all_data 文件夹下；
   然后读取 all_data 中所有图片，生成数据集，分别进行以下特征提取，并将得到的 3 组特征进行融合，并使用 joblib 保存：
   a) 计算 SIFT 特征，并用 K-means 和词袋模型进行特征提取（向低维映射）
   b) 计算 HOG 特征，并用 PCA 算法进行特征提取（向低维映射）
   c) 计算 LBP 特征，并用直方图和 PCA 算法进行特征提取（向低维映射）
2. 读取保存的特征，进行 K 折交叉验证中，使用选出的 4 个分类器集成生成模型，记录模型正确率衡量指标变化，并生成在 test 集上的预测标签存入 csv 文件用于提交。

程序共有以下 7 个模块 py 文件、3 个文件夹，各文件及文件夹功能如下：

文件夹：

1. data：从 Kaggle 官网上下载的原始数据集，其下有子目录 test（存有所有测试集图片）、子目录 train（存有所有训练集图片，图片按照其种类处于其物种名所命名的文件夹下）；
2. all_data：所有经过预处理的图片所在文件夹，其结构、子文件命名、文件命名和原始数据集 data 完全一致，由 Image_Process.py 运行生成；
3. res：程序运行时所有中间数据（提取后的各类特征）的保存位置，由 mian.py、SIFT.py、HOG.py、LBP.py 等文件生成。

文件：

1. main.py：运行 main.py 以完成从最开始的图像预处理，到图像特征提取和融合，到分类器训练和集成，到最终的 K 折交叉验证正确率的整个流程，将调用到和依赖于其他所有模块；
2. Image_Process.py：对图像进行预处理。该程序将会从 data 文件夹读取图片，将图片进行均衡化、根据颜色分割图片的操作，并将完成后的所有图片保存至 all_data 文件夹。可单独执行；
3. Data_Load.py：从指定文件夹读取下的 train、test 文件夹内所有图片，并返回三个 numpy 数组 train_dataset、train_label、test_dataset，依次为：train 训练集所有图片、train 训练集所有标签、test 测试集所有图片；
4. Defines.py：定义 K 折交叉验证的 K 数 n_splits 等各类超参、PCA 降维等功能函数、生成并训练单独一个分类器如 Clf_SVC 等分类器生成函数。不可单独执行；
5. SIFT.py：使用 SIFT+词袋模型，提取和转换图像特征的模块。可单独执行以使用仅 SIFT 特征和单独 SVC 分类器的测试；
6. HOG.py：使用 HOG 和 PCA 降维，提取和转换图像特征的模块。可单独执行以使用仅 HOG 特征和单独 SVC 分类器的测试；
7. LBP.py：使用 LBP 和 PCA 降维，提取和转换图像特征的模块。可单独执行以使用仅 LBP 特征和单独 SVC 分类器的测试。

# 二、图片预处理

图像预处理部分，将会从 data 文件夹读取图片，将图片进行均衡化、根据颜色分割图片的操作，并将处理完成后的所有图片保存至 all_data 文件夹。程序整体使用彩色 RGB 三通道图像，所以该模块读取和保存的都是彩色 RGB 三通道图像。

## 1、均衡化的实现

使用 OpenCV2 实现图像的均衡化。步骤如下：

1. 使用 cv2.split()函数，将图像的三个颜色通道分裂，分别为 B、G、R（OpenCV2 中通道顺序与正常 RGB 顺序相反）；
2. 使用 cv2. equalizeHist ()函数，将图像的三个颜色通道分别进行均衡化处理；
3. 使用 cv2. merge ()函数，将处理后的图像的三个颜色通道合并，保持(B, G, R)的最初顺序。

对原始数据集图片直接进行均衡化效果与原图的对比如下图 2.1 所示：


图 2.1　对原始数据集图片直接进行均衡化

可见，对于该数据集，均衡化后的图片植物幼苗部分的绿色更加突出，且将背景中原本的黄色、棕色等与绿色在 HSV 空间颜色表上相近的颜色向远离绿色的方向变化为偏红色、蓝色的颜色，因此有助于后续的图像分割操作。

实现代码如下：

```
(B, G, R) = cv2.split(image)
B = cv2.equalizeHist(B)
G = cv2.equalizeHist(G)
R = cv2.equalizeHist(R)
result = cv2.merge((B, G, R))
```

## 2、图像锐化的实现

使用 OpenCV2 实现图像的锐化。步骤如下：

1. 实现图像锐化的算子有很多，本实验采用拉普拉斯算子进行锐化，定义的拉普拉斯算子如下：

$$sharpen\_op = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

2. 使用 cv2. filter2D ()函数，用算子对图像进行遍历，得到计算后的结果；
3. 因为使用算子后，图片中的像素处的值可能超出[0, 255]的范围，所以再使用 cv2. convertScaleAbs ()函数，将第二步得到的结果转换到[0, 255]的范围内。

对原始数据集图片直接进行锐化效果与原图的对比如下图 2.2 所示：

图 2.2 对原始数据集图片直接进行锐化

可见，对于该数据集，锐化后的图片植物幼苗部分如其脉络、边缘变得更加清晰，但同时叶片上的噪点，也有较明显的增加。最终实验发现，若增加图像锐化操作，在特征提取时如 SIFT 提取出的特征将会大幅增加，但实际最终却会导致图像分类的正确率总体下降，因此，最终的方案中未使用锐化操作。这将在后文中进行详细描述。

实现代码如下：

```
"""对图像使用拉普拉斯算子进行锐化"""
sharpen_op = numpy.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], 'float32')
result = cv2.filter2D(image, cv2.CV_32F, sharpen_op)
result = cv2.convertScaleAbs(result)
```

## 3、根据颜色进行图像分割的实现

因为本实验数据集所有图片中，只有植物幼苗部分呈现绿色，背景中其他所有颜色都为黄色、棕色、白色、黑色等其他非绿色的颜色，所有很适合使用颜色范围来对图像进行分割。

根据颜色分割一般都将图片转换到 HSV 空间进行处理，其中：H 为颜色，S 为饱和度，V 为亮度。所以，使用 OpenCV2 实现根据颜色的图像分割。步骤如下：

1. “mask”为进行分割的蒙板，cv2.GaussianBlur()函数先将模板模糊一下，以缓解根据颜色进行图像分割时，由于植物幼苗边缘处颜色的不连续变化导致出现的锯齿现象；
2. 使用 cv2.cvtColor(mask, cv2.COLOR_BGR2HSV)将蒙板 mask 转到 HSV 空间进行颜色分割；
3. 通过 cv2.inRange()函数将 mask 转换为只有绿色部分为 1（True），其他部分均为 0（False）的蒙板；
4. image 为需要分割的图像。最后通过 cv2.bitwise_and(image, image, mask=mask)，用前三步中得到的蒙板 mask 对 image 进行分割，即保留用蒙板记录着的要保留的属于绿色区间[green_lower, green_upper]内的部分，其他部分置为 0。

资料显示，在 HSV 空间各颜色范围大致如下图 2.3.1 所示：

| | 黑 | 灰 | 白 | 红 | | 橙 | 黄 | 绿 | 青 | 蓝 | 紫 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| hmin | 0 | 0 | 0 | 0 | 156 | 11 | 26 | 35 | 78 | 100 | 125 |
| hmax | 180 | 180 | 180 | 10 | 180 | 25 | 34 ; | 77 | 99 | 124 | 155 |
| smin | 0 | 0 | 0 | 43 | | 43 | 43 | 43 | 43 | 43 | 43 |
| smax | 255 | 43 | 30 | 255 | | 255 | 255 | 255 | 255 | 255 | 255 |
| vmin | 0 | 46 | 221 | 46 | | 46 | 46 | 46 | 46 | 46 | 46 |
| vmax | 46 | 220 | 255 | 255 | | 255 | 255 | 255 | 255 | 255 | 255 |

图 2.3.1 HSV 空间各颜色范围

即在 HSV 空间，绿色的范围普遍界定为：

$$绿色下限 green\_lower = [35, 43, 46]$$

$$绿色上限 green\_upper = [77, 255, 255]$$

记为区间 1。

而经过测试，发现以下区间能更好地将本实验中的植物幼苗部分完整地划分出来：

$$绿色下限green\_lower = [29, 43, 46]$$

$$绿色上限green\_upper = [77, 255, 255]$$

记为区间 2。

对原始数据集图片直接根据颜色进行图像分割时，两种范围设定下图像分割效果如下图 2.3.2 所示，左边为使用区间 1 的效果，右边为使用区间 2 的效果：

图 2.3.2　两种范围设定下图像分割效果对比

可见，区间 2 极好地实现了根据颜色进行图像分割的目的，且分割出的图像完整地保留着原图中的所有植物幼苗部分，较区间 1 有较明显地提升，因此，在最终的方案中，采用区间 2 进行图像分割。

实现代码如下：

```
# 绿色的HSV范围：色调-饱和度-明度
green_lower = numpy.array([29, 43, 46])  # 绿色下限为[35, 43, 46]
green_upper = numpy.array([77, 255, 255])  # 绿色上限为[77, 255, 255]

"""将图像转换到HSV空间以筛选出图像中的绿色区域，其他区域置为0"""
# 先将mask模糊
mask = cv2.GaussianBlur(mask, (3, 3), 0)
# mask转到HSV空间进行颜色分割
mask = cv2.cvtColor(mask, cv2.COLOR_BGR2HSV)
# 根据阈值找到对应颜色
mask = cv2.inRange(mask, green_lower, green_upper)
result = cv2.bitwise_and(image, image, mask=mask)
```

## 4、处理方法及其顺序选取

为研究均衡化、锐化、分割三种操作的选取及其作用顺序不同，对实验准确率的影响情况，做了以下探究实验。综合考虑到数据集十分庞大，对完整的数据集进行一次处理、特征提取、分类器训练、验证集准确率获取需要数分钟时间。

而通过实验观察到，当训练集减小时，虽然准确率固然会出现明显下降，但下降后因参数设置而导致的不同实验间的正确率相对大小，仍然与在完整数据集上的正确率相对大小情况相同。即在规模减小后的数据集上，仍能正确反映出不同参数设置导致的模型正确率变化情况。

所以在探究实验中，使用以下设置缩短单次实验所需时间：

1. 使用的 train 训练集中，每一类物种的图片数量都限制为 50 张（12 类，共 600 张图片），位于“small_data”文件夹下；
2. SIFT+词袋模型提取出的特征用于训练分类器时的准确率是三种中单独使用时最高的，对模型整体准确率的提高作用最为显著，所以单独使用 SIFT 特征进行探究实验，以减少程序一次运行所需时间；
3. 仅使用随机森林一种分类器进行训练，以减少程序一次运行所需时间。

探究实验中，使用的参数设置如下：

- 每一类物种的图片数量：50 张图
- 特征提取：SIFT+词袋模型，词袋大小为：50
- 分类器：随机森林（所有参数均为默认值未调整）

对以下 6 组不同的处理方法选取及其顺序设置进行了实验：

- 正则化、再去除背景
- 去除背景、再正则化
- 正则化、再去除背景、再锐化
- 锐化、再去除背景、再正则化
- 去背景、再锐化、再正则化
- 去除背景、再锐化

每组设置均进行了 3~4 次的实验，共 19 次。各次实验的实验结果（精确率、召回率、F1 评价指标得分）记录在附件 1《附录 1：对于图片预处理方法选择及顺序的考察.docx》中。

实验中发现若加入锐化反而会导致正确率下降和方差增大。“正则化、再去除背景”与“正则化、再去除背景、再锐化”设置的图片处理结果对比如下图 2.4 所示：

图 2.4　预处理是否加入锐化时的效果对比

可见，对于该数据集，锐化后的图片植物幼苗部分变得更加清晰如其脉络、边缘；但同时，叶片上的噪点，也有较明显的增加。

实验中观测到，若增加图像锐化操作，在特征提取时如 SIFT 提取出的特征将会大幅增加：

- 600 张图片“正则化、再去除背景”设置，检测出的 SIFT 特征点总数：125604
- 600 张图片“正则化、再去除背景、再锐化”设置，检测出的 SIFT 特征点总数：178693

但实际最终却会导致图像分类的平均正确率总体下降和方差增大，因此最终的方案中未使用锐化操作。

综合评价各种设置下的正确率及稳定性，其中使用“正则化、再去除背景”设置的 3 次实验的平均正确率及方差在所有 6 组设置中取得了最好的效果，所以最终图片预处理采用的操作及其处理顺序为：先正则化、再去除背景。

# 三、特征提取：SIFT+词袋模型

计算和提取 SIFT 特征在 SIFT.py 文件中完成，主要分为 3 步：

1. 计算出各图片的 SIFT 特征（SIFT.py 文件中的 Sift 函数）；
2. 然后将所有特征使用 K 聚类算法找到指定的 SIFT_BOW_size 个聚类中心，即为词袋模型中的词典（所有单词），词袋大小即为 SIFT_BOW_size（文件中的 Kmeans 函数）；
3. 最后依据各图片的 SIFT 特征、词典，计算“单词文档共现矩阵”。即一张图片看作一篇“文章”，含有若干个“句子”（特征看作一个“句子”），每个句子含有若干“词汇”（cluster.vq.vq 计算出的对聚类中心的属于关系）；进而可求出 TF、IDF，并用单词文档共现矩阵*IDF 表示图片的最终的特征（文件中的 BOW 函数）。

## 1、计算 SIFT 特征

**过程：**

使用 OpenCV2 中的 xfeatures2d 模块进行 SIFT 特征计算，对于每一张图片 image：

1. 首先通过 cv2.xfeatures2d.SIFT_create()实例化一个可以计算 SIFT 特征点的对象 detector
2. 然后使用 detector 的 detectAndCompute(image, None)函数，其将完成对图像 image 中的 SIFT 特征点检测功能；
3. 其返回值依次为找到的所有关键点 kps、关键点对应计算出的 SIFT 特征（每一个关键点的特征都是 128 维），将改图的 SIFT 特征加入到 sift_list

**未找到特征点情况处理：**

对于每一张图片，由于 OpenCV2 的 SIFT 计算找到特征点数量是不确定的，因此每一张图片得到的 SIFT 特征数量也是不同的，其维度可表示为[x, 128]，其中 x 为未知非负整数。

如下图 3.1.1 所示，未经过图像分割和经过图像分割的图片，其关键点分布与数量就存在着极大的差异：

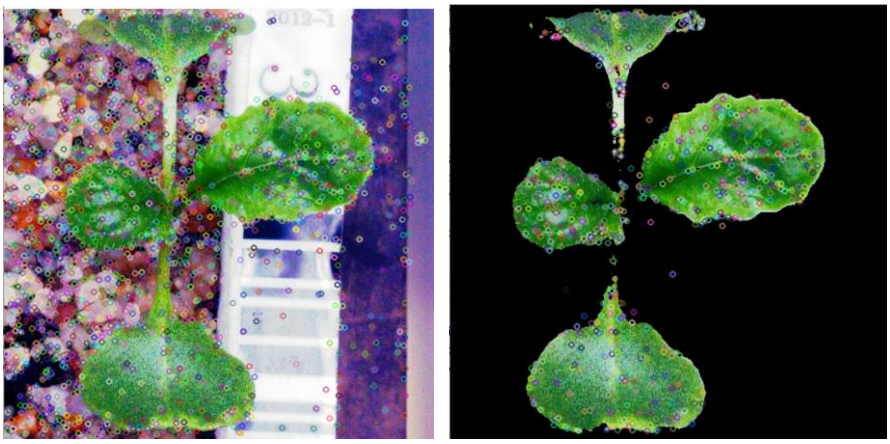

图 3.1.1 SIFT 检测到的关键点数量不同

而当一张图片未找到特征点时，实例化对象 detector 会返回 None，显然，None 若不经处理，在后续的将 SIFT 特征转换为 numpy 数组、和 K 聚类等过程中就会导致错误。

提出以下三种方案：

1. 方案 1：直接将该图片的特征不加入 sift_list。但这样会导致得到的 SIFT 特征列表 sift_list 和标签列表 train_label 数量不对应，如此会导致在后续的分类器训练和验证集划分时出现错误。所以该方案不可取；
2. 方案 2：将未找到特征点的图像填充 0 向量到 sift_list 中。这样做可以解决方案 1 中的 SIFT 特征列表 sift_list 和标签列表 train_label 数量不对应问题，但结果是所有未检测出关键点的图片在最终 SIFT 特征均为一个相同的不确定的值（类似“均值”），及所有未检测出关键点的图片最终会被随机分到一组固定的类中，势必会影响图片分类的准确率。
3. 方案 3：将未找到特征点的图像填充上一个图像的 SIFT 特征向量到 sift_list 中。这样做也就默认该图片与上一张种类相同，考虑到提取特征时，图像处理顺序均为按原文件夹依次排列的，所以在这本实验的情况下，在一定程度上有利于在验证集上模型准确率的提高。

最终采用方案 3。

该过程位于 SIFT.py 文件中的 Sift()函数，主要代码如下：

```
sift_list = []
detector = cv2.xfeatures2d.SIFT_create()
for image in tqdm(dataset):
    kp, sift = detector.detectAndCompute(image, None)  # des 是描述子 sift 是关键点特征，每张图关键点不一样多，每个关键点是 128 维
    if sift is None:
        sift = sift_list[len(sift_list) - 1]
    sift_list.append(sift)
```

该过程示例图如下图 3.1.2 所示：

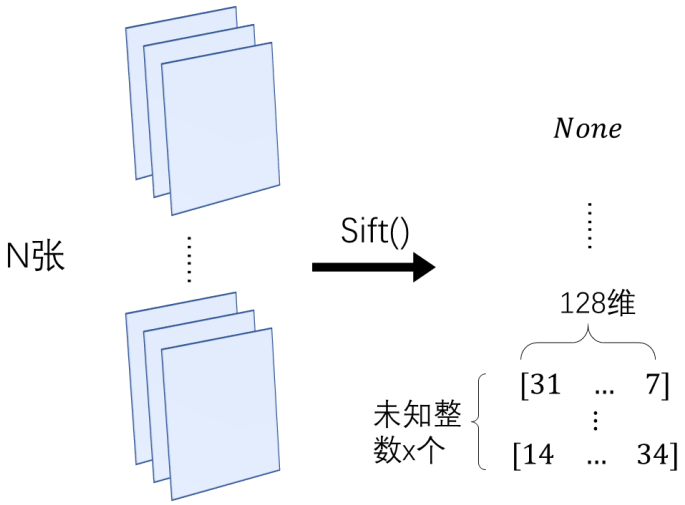


图 3.1.2　Sift 函数过程示例图

## 2、K-means 聚类构造字典

使用 scipy.cluster 模块中的 vq.kmeans()函数进行 K 聚类，步骤如下：

在 Sift 函数中得到的所有图片的 SIFT 特征 sift_list 为 list 列表，通过 numpy.vstack()将所有图片的所有特征连接为一个二维矩阵 descriptors 中，对于本实验，所有 5544 张图片共有 972183 个 SIFT 特征点，平均 175 个/张；

然后使用 cluster.vq.kmeans()函数将 descriptors 进行聚类，指定其聚类中心个数为 WOB_size（本实验中设置为 80）；

函数返回值为图片是否划分为该聚类中心 cluster_dict（是一个[5544, WOB_size]的二维仅含 0 和 1 的矩阵，即每张图片对应一个长度为 50 的一维向量）、损失 distortion（与各聚类中心的欧氏距离）。

## 3、词袋模型提取特征

依据各图片的 SIFT 特征、和在上一个步骤 2 中得到的词典（聚类中心）cluster_dict，计算“单词文档共现矩阵”。即一张图片看作一篇“文章”，含有若干个“句子”（特征看作一个“句子”），每个句子含有若干“词汇”：

1. 用 cluster.vq.vq 计算出一张图片的特征对聚类中心的属于关系，构造单词文档共现矩阵 im_features，其维度为[n, BOW_size]（n 为图片数量）；
2. 使用 numpy.sum()函数对 im_features 进而每列的纵向求和即可求出 TF 矩阵，其维度为[BOW_size, ]；
3. IDF 的计算公式如下，分子分母可分别加 1，用 log(x+1)防止 x 为 0 导致数值错误；

$$IDF = log(文档总数N / TF)$$

用单词文档共现矩阵 im_features * IDF，即可用于表示图片的最终的特征。

该过程示例图如下图 3.3 所示：

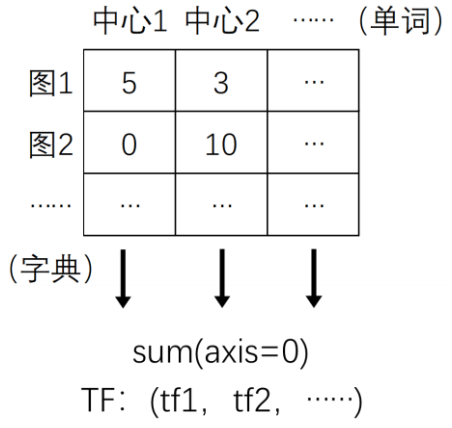


图 3.3 BOW()函数过程示例图

# 四、特征提取：HOG 特征和 LBP 特征

## 1、HOG 特征

选择原因：

HOG(Histogram of Oriented Gradient）是一种描述方向梯度直方特征的算子。通过计算与统计图像局部区域的梯度方向直方图来构成特征。边缘是图像颜色剧变的区域，在一副图像中，局部目标的表象与形状能够被梯度或边缘的方向密度分布很好地描述。

在植物幼苗的分类中，HOG 算子提取的是植物叶片边缘的特征，显然可以依靠植物叶片的轮廓来判断植物叶片的种类。

实现原理：

HOG 特征提取实现方法是将图像分成小的 cell 连通区域，将每个 Cell 里面的梯度幅值按照 9 个方向进行统计。计算完之后，将会产生一个横坐标 X 为梯度方向，纵坐标 Y 为梯度幅值的方向梯度直方图。然后采集细胞单元中各像素点的梯度的或边缘的方向直方图,最后把这些直方图组合起来形成 HOG 特征描述器。把这些局部直方图在图像的更大的 block 范围内进行归一化，防止光照不均匀的变化形成对比差异。

代码实现：

在实现 HOG 算法时，通过调用 skimage.feature 函数库中的 hog 函数进行提取 HOG 特征。

首先，将彩色图使用 cv2.cvtColor 转化为灰度图 gray（提取边梯度特征，灰度图即可）；在调用 hog 函数时，传入 gray 灰度图片后，将方向 orientations 设置为 6，连通区域 cell 设置为 20*20，收集 cell 特征到，连通区域为 2*2 的 block；最后收集 block 中的特征形成 HOG 特征。返回的是每个图片的特征（一维的），在该实验中，一张图片提取的特征为 48600 个元素，因此通过 PCA 算法进行降维。

## 2、LBP 特征

选择原因：

LBP（Local Binary Pattern）是一种用来描述图像局部纹理特征的算子，用于提取图像的局部纹理特征，具有旋转不变性和灰度不变性等显著的优点，可以用于描述植物幼苗的纹理特征。

每种幼苗叶片表面的叶脉、叶茎具有不同的纹理特征，在现实生活中也常常根据叶片的这些纹理特征进行植物种类的分类，所以，选择 LBP 算子作为训练特征是有显示依据

的，LBP 可以在纹理方面良好的体现叶片的特征。

实现原理：

在一个 3*3 的窗口内，以窗口中心像素为阈值，与相邻的 8 个像素的灰度值比较，若周围的像素值大于中心像素值，则该位置被标记为 1;，否则标记为 0。如此可以得到一个 8 位二进制数，并将其转换为共 256 种的 10 进制的 LBP 码，将这个 LBP 码作为窗口中心像素点的 LBP 值，以此来反映这个 3*3 区域的纹理信息。

代码实现：

在实现 LBP 算法时，通过调用 skimage.feature 函数库中的 local_binary _pattern()函数进行提取 LBP 特征。首先，将彩色图使用 cv2.cvtColor 转化为灰度图 gray（提取纹理特征，只需要灰度图可），在调用 local_binary_pattern 函数时，传入 gray 灰度图片，并将参数中心 radius 设置为 1，范围设置为常规的 8 个（3*3 区域内）。返回的结果 lbp 为与原来图片尺寸相同（[256, 256]）的二维特征，后续先使用直方图处理降为一维，再使用 PCA 算法降维。

## 2、HOG 与 LBP 使用的降维方式：直方图、PCA

经过 HOG、LBP 提取的特征，会出现特征维度不匹配、特征过多、噪音过多或者过拟合等现象。所以，需要进一步降维处理和关键特征提取。

HOG 提取出的特征，每一张图片提取出的特征维度为：(48600,)。因此采用 PCA 算法进行降维。

LBP 提取出的特征与原来图片尺寸相同（[256, 256]）的二维特征，先使用直方图统计处理降为一维，再使用 PCA 算法降维。

首先使用是直方图，将大特征，或者二维的特征，收缩为一个一维的特征。使用 numpy 函数库中的 histogram 函数，其中参数设置，density 为 True 结果为 bin 处的概率密度函数的值,宽度 bins 为最大特征数，浮动范围 range 为 0 到 bins。

在使用直方图后，特征虽然是减少到 2000 左右，但是特征仍然是过多的，有噪音特征混杂，需要进一步的降维处理。

使用 sklearn.decomposition 函数库中的 PCA 函数。PCA 函数可将高维特征映射到低维上，并选取各特征之间的方差最大，协方差为 0 的关键特征。使用 PCA 可以设置降维后，特征的个数，或者是特征的权重。

经过实验发现，模型整体得到的最好的结果时，LBP 降维后参数个数为 60，HOG 降维后参数个数为 80，SIFT 特征个数为 80。

# 五、分类器与特征融合

## 1、分类器选择

### 1. 支持向量机分类器（SVM）

选择原因：

在本次实验中支持向量机中 SVC。该分类器常用于多元分类回归，具有良好的预测生成器，提供了鲁棒的过拟合、噪声数据和异常点处理，训练快速，自动检测数据的非线性等优点。

代码实现：

使用 sklearn.svm 函数库中的 SVC 函数训练。

参数解释：

kernel 核函数选择为默认的高斯核‘rbf’，gamma 核系数选择‘scale’，probability 是指启动概率估计，C 为正则化参数，设置为 12。

### 2. 随机森林分类器

选择原因：

随机森林是基于多颗决策树的一种集成学习算法，其输出的类别是由多个决策树输出的类别的众数。随机森林是 bagging 算法的一种，bagging 算法是指随机抽取 n 个训练样本，共进行 k 轮抽取，得到 k 个训练集，对于 k 个训练集，训练 k 个决策树模型，最后由多棵决策树进行投票表决，产生最终的分类结果。随机森林使用多种决策树进行并行运算，最后，将多个决策树的分类结果汇总形成一个更加良好体现分类标签的结果，同时在训练不同决策树时，本身也进行了一种交叉验证，增强了实验的准确性。

代码实现：

调用 sklearn.ensemble 函数库中的 RandomForestClassifier 函数。

参数解释：

决策树个数 n_estimators=400,最佳分割时的功能数量 max_features='sqrt'；

树的最大深度 max_depth=40，袋外样本估计泛化精度 oob_score，随机种子 random_state=10。

### 3. ExtraTree 分类器集合

选择原因：

ExtraTree 中文名为极端选择树，与随机森林算法十分相似，都是由许多决策树构成。但是不同的是，ExtraTree 使用的所有的样本，只是特征是随机选取的，因为分裂是

随机的，所以能够更加随机地体现分类的结果，避免过拟合和噪音污染，在某种程度上比随机森林得到的结果更加好。在本题中，分别尝试了随机森林和 ExtraTree 算法，其单独的效果相差不大，但是在综合进行集成学习后，准确率有了很大的提高

代码实现：

调用 sklearn.ensemble 函数库中的 ExtraTreesClassifier 函数。

参数解释：

决策树个数 n_estimators=300,最佳分割时的功能数量 max_features='sqrt'；

树的最大深度 max_depth=50。

**4. XGBoost 分类器**

选择原因：

XGBoost 是一种迭代的决策树算法，该算法由多棵决策树组成。XGBoost 是在 GBDT 的基础上对 boosting 算法进行的改进，内部决策树使用的是回归树。所有树的结论累加起来做最终答案。算法泛化能力（generalization)较强。训练结果精度高。

代码实现：

调用 xgboost 函数库的 XGBClassifier 函数。

参数解释：

learning_rate 学习率；

n_estimators 迭代次数；

max_depth 指定树的最大深度，防止过拟合。

**5. KNN 分类器**

选择原因：

KNN（K-Nearest Neighbor）最邻近分类算法以所有已知类别的样本作为参照，计算未知样本与所有已知样本的距离，从中选取与未知样本距离最近的 K 个已知样本，根据少数服从多数的投票法则，将未知样本与 K 个最邻近样本中所属类别占比较多的归为一类。实验发现，在尝试该分类器后，发现准确率在 0.7 左右，模型整体准确率下降，因此 KNN 分类器被舍弃，没有被参与最终的集成。

代码实现：

调用 sklearn.neighbors 函数库的 KNeighborsClassifier 函数。

参数解释：

n_neighbors 近邻数 weights 预测中使用的权重函数。

**6. 弹性网络（ElasticNet）**

选择原因：

在线性回归的基础上加上了一个 L1,L2 惩罚项。这样的组合既学习了一个稀疏的模型

(类似 Lasso),同时也保持了岭回归的正则属性。当多个特征和另一个特征相关的时候弹性网络非常有用。Lasso 倾向于随机选择其中一个，而弹性网络更倾向于选择两个。该模型在训练时，只有少量参数稀疏，可以产生有效解多，在循环迭代中比较稳定。但是训练效果不好，被舍弃。

参数解释：

l1_ratio：对于 l1_ratio=0，惩罚为 L2；l1_ratio=1，为 L1 范数。对于 0<l1_ratio<1，惩罚是 L1 和 L2 的组合；

max_iter：指定最大迭代数；

normalize：一个布尔值。如果为 True，那么训练样本会在回归之前会被归一化；

random_state：随机数生成器的种子。

## 2、分类器集成

通过实验观测，共选取选取了 SVC 支持向量机，随机森林分类器，ExtraTree 分类器，XGBoost 分类器这四个分类器的集成学习。不同的分类器对同一个图片的 12 种标签预测有不同的概率，将四个分类器对同一张图片的 12 种标签的概率相加后，选取 12 种标签中概率最大的标签作为该图片预测结果的标签。在集成学习中，调用各分类器的 predict_proba() 函数即可获得各分类器对某图片属于各类的概率。

集成学习有助于增加分类的准确性，当某种分类器关注于一个错误特征而将其分类错误时，其他分类器可能都会避免这个错误，而集成学习后，错误标签的概率下降，正确标签的概率提高，从而选择出更正确的结果。通过集成学习增加了分类的准确性。

其实现代码如下所示：

```
cls_list = list()
cls_list.append(Clf_SVC(train_data, train_label))
cls_list.append(Clf_RandomForestClassifier(train_data, train_label))
cls_list.append(Clf_ExtraTreesClassifier(train_data, train_label))
cls_list.append(Cls_XGBooster(train_data, train_label))

probability = cls_list[0].predict_proba(data_inputs)
for cls in cls_list[1:]:
    probability = np.add(probability, cls.predict_proba(data_inputs))
# 取出最大的概率作为其标签
predict = np.argmax(probability, axis=1)
```

### 3、特征融合

选择了将进行关键特征提取后的 SIFT，LBP，HOG 三个特征进行横向连接，来作为一张图片的特征。生成后的新特征能从纹理，轮廓以及 SIFT 角度上，良好地反映一张图片的信息。单独使用 SIFT，LBP，HOG 特征时，都是一些简单的特征，只能够简单地从图片的某一特征进行判断。

实验中测得，在单独使用 SIFT 特征时，f1 评分为 0.727；单独使用 HOG 特征时，f1 评分为 0.697；单独使用 LBP 时，f1 评分为 0.689。但是，在使用特征融合后，实验的 f1 评分达到了 0.865，体现了特征融合的有效性和必要性。

## 六、数据读取、test 集处理、K 折交叉验证

### 1、数据读取

**train 训练集：**

train 训练集的读取使用 torchvision.datasets 模块中的 ImageFolder，其使用的 transforms 仅对图像进行缩放处理，本实验中图片尺寸设定统一大小为[256, 256]：

```
transforms = torchvision.transforms.Compose([
    # 缩放图像以创建 image_size x image_size 的新图像
    torchvision.transforms.Resize((image_size, image_size))])
```

通过 ImageFolder 即可以 PIL 库中 Image 格式获取所有图片，而后续程序提取特征需要使用的为 numpy 型数据，因此，通过以下语句将读取的所有图片转化为 numpy 数组：

```
train_dataset = numpy.array([numpy.array(image) for image, _ in tqdm(dataset)])
train_label = numpy.array([label for _, label in tqdm(dataset)])
```

完成后的 train_dataset 即保存有所有训练集中的图片数据，其维度含义依次为[图片数量，图片宽度，图片高度，图片通道数量]，在本实验中其至依值次为[4750, 256, 256, 3]；train_label 即保存有所有训练集中的图片的标签，其为一位数组，与 train_dataset 中图片的标签一一对应。标签的范围为 0~11，对应关系如下：

0: 'Black-grass'、 1: 'Charlock'、 2: 'Cleavers'、 3: 'Common Chickweed'、 4: 'Common wheat'、 5: 'Fat Hen'、 6: 'Loose Silky-bent'、 7: 'Maize'、 8: 'Scentless Mayweed'、 9: 'Shepherds Purse'、 10: 'Small-flowered Cranesbill'、 11: 'Sugar beet'

**test 测试集：**

test 测试集中图片都位于同一级路径，因此使用 files = os.listdir('data/test')语句即可获得 tes 测试集中所有图片名，然后 PIL 的 Image 模块对图片依次进行读取、裁剪、添加

到保存有所有测试集中的图片数据的 test_dataset 中即可，其维度含义依次为[图片数量，图片宽度，图片高度，图片通道数量]，在本实验中其至依值次为[794, 256, 256, 3]。

## 2、test 集并入处理

因为程序在特征提取过程中涉及到 K 聚类、PCA 降维等操作，所以应该将 test 验证集在特征提取时与 train 训练集一同进行特征提取。则需要在特征提取前将 test 集图片数据与 train 集图片数据进行连接，将其二者一起进行 SIFT、HOG、LBP 的特征提取进程。

在本程序中，通过修改启动 SIFT、HOG、LBP 特征提取进程的函数即可：

1. 开始前使用 concatenate()函数将 test 集图片数据与 train 集图片数据连接，合并为 all_image；
2. 完成后将 all_image 再拆分为原 test 和 train 对应数据，再进行保存。

以 HOG 特征提取为例，实现代码如下：

```
# 将 train 和 test 集合并到一起进行特征提取（train, test）
all_image = numpy.concatenate((train_data, test_data))
all_features = HOG_detect(all_image)
# 保存时，将 train 和 test 集合分开
joblib.dump((all_features[:len(train_data)], train_label,
all_features[len(train_data):]),
          f'res/hog_features_pca={HOG_PCA_size}.pkl')
```

## 3、K 折交叉验证

为检验程序的准确性，采用 K 折交叉验证（K=5）来将 train 划分为训练集和验证集。

1. 对于已获取到的 train 集上所有图片的特征数组 train_features，使用 sklearn.model_selection 模块中的 KFold()函数，将其随机划分为 K 折
2. 对于每一折，都使用划分得到的 train_X 训练全新的一组分类器；
3. 并用该组全新的分类器在当前折中的 val_X 验证集计算正确率等衡量指标，和生成 test 集的预测标签。

```
kfold = KFold(n_splits=n_splits, shuffle=True, random_state=46)
for train_index, val_index in kfold.split(train_features):
    train_X, val_X = train_features[train_index],
train_features[val_index]
    train_Y, val_Y = train_labels[train_index], train_labels[val_index]
    cls_list = Step2_Train_Classification(train_X, train_Y)
```

# 七、实验结果

## 1、validation 验证集指标

使用 K 折交叉验证（K=5）时，模型在 validation 验证集上的各指标情况如图 7.1 所示：

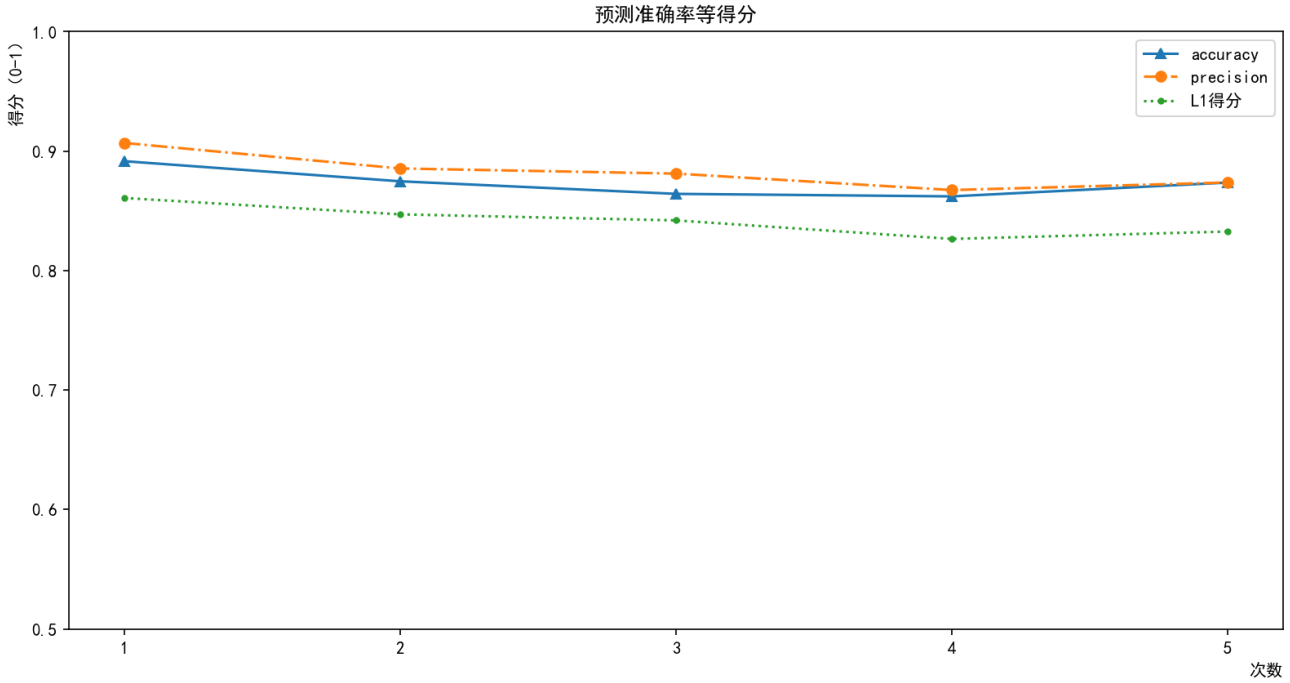


图 7.1　K 折交叉验证中模型在验证集上的各指标变化情况

该次测试中，选用的分类器如下：

SVC(C=12, probability=True)

RandomForestClassifier(bootstrap=False, max_depth=40, max_features='sqrt',
n_estimators=400, random_state=10)

ExtraTreesClassifier(max_depth=50, max_features='sqrt', n_estimators=400)

XGBClassifier(learning_rate=0.13, max_delta_step=None, max_depth=5,
n_estimators=300, n_jobs=None, nthread=10, num_parallel_tree=None ...)

K 折交叉验证中（K=5），五次详细数据如下五个框图中所示。其中，正确率最高可达 89.16%，精确率最高可达 90.68%，召回率最高可达 84.92%，F1 得分最高可达 86.075%。

正确个数：847,总数：950，正确率 0.891579
m_precision:　0.9068481986304527
m_recall:　0.849232336867204
f1-score:　0.8607578872109566

| | |
|---|---|
| 正确个数:831,总数:950, 正确率 0.874737<br>m_precision: 0.8854841348357777<br>m_recall: 0.8401500831876186<br>f1-score: 0.8470711004514557 | 正确个数:821,总数:950, 正确率 0.864211<br>m_precision: 0.881186910467686<br>m_recall: 0.8351527708899132<br>f1-score: 0.8420068403999151 |
| 正确个数:819,总数:950, 正确率 0.862105<br>m_precision: 0.8674057876929618<br>m_recall: 0.8131302859095274<br>f1-score: 0.826502332281113 | 正确个数:830,总数:950, 正确率 0.873684<br>m_precision: 0.8737601750601418<br>m_recall: 0.8219568087300729<br>f1-score: 0.8326793470898682 |

其中：

| | 预测值=1 | 预测值=0 |
|---|---|---|
| 真实值=1 | TP | FN |
| 真实值=0 | FP | TN |

正确率：$accuracy = (TP + TN)/(TP + TN + FP + FN)$

精确率：$precision = TP/(TP + FP)$

召回率：$recall = TP/(TP + FN)$

L1 得分：$L1\_score = (2 * precision * recall)/(precision + recall)$

## 2、test 测试集指标

使用生成的 csv 文件提交至 Kaggle 后，测得在验证集上的 F1-score 为 0.75755，如下图 7.2 所示：

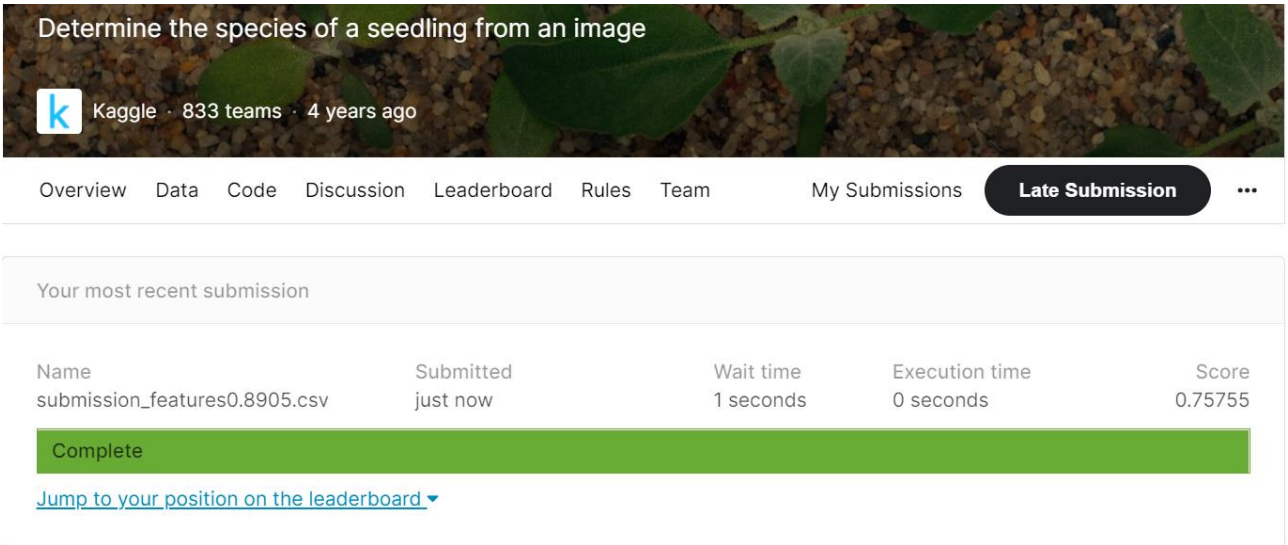


图 7.2　Kaggle 得分截图

# 附录

## 附录 1

参见文档《附录 1：对于图片预处理方法选择及顺序的考察.docx》

## 源码

### 1. main.py

```
from Defines import *
from Image_Process import Process_All, is_equalize, is_sharpening

from SIFT import Train_Sift
from HOG import Train_HOG
from LBP import Train_LBP

from sklearn.model_selection import KFold


def Step1_Get_Feature(is_preload=is_preload):
    """获取并计算 train 特征、train 标签、test 特征（特征为 SIFT、HOG、LBP 特征融合）
    is_preload=True 则从已保存的文件直接读取特征"""
    # 如果 is_preload=False，重新计算 SIFT、HOG、LBP 特征并保存
    if not is_preload:
        # 没有 all_data 文件夹则先进行一下图片预处理
        if not os.path.exists(path):
            Process_All(file_from='data', file_to='all_', number_limit=None,
                        is_sharpening=is_sharpening, is_equalize=is_equalize,
is_display=False)
        train_data, train_label, test_data = Data_Load(path)
        # 计算所有特征并保存（train、test 共同进行特征提取）
        Train_Sift(train_data, train_label, test_data)
        Train_HOG(train_data, train_label, test_data)
        Train_LBP(train_data, train_label, test_data)
    # 读取所有 train、test 集合图片的特征
    sift_train_data, train_label, sift_test_data =
joblib.load(f'res/sift_features_bow={SIFT_BOW_size}.pkl')
    hog_train_data, _, hog_test_data =
joblib.load(f'res/hog_features_pca={HOG_PCA_size}.pkl')
    lbp_train_data, _, lbp_test_data =
joblib.load(f'res/lbp_features_pca={LBP_PCA_size}.pkl')

    # 特征融合
    train_features = merge(sift_train_data, hog_train_data, lbp_train_data)
    test_features = merge(sift_test_data, hog_test_data, lbp_test_data)
    return train_features, train_label, test_features
```

```
def Step2_Train_Classification(train_data, train_label):
    """选取多个分类器，并全部用输入的(data, label)训练好
    返回列表包含任意个训练好的分类器"""
    # 选取并训练分类器
    cls_list = list()
    cls list.append(Clf SVC(train data, train label))
    cls_list.append(Clf_RandomForestClassifier(train_data, train_label))
    cls_list.append(Clf_ExtraTreesClassifier(train_data, train_label))
    cls list.append(Cls XGBooster(train data, train label))
    return cls_list


def Step3_Predict(data_inputs, cls_list):
    """用传入的列表中的多个分类器，依次对传入的数据进行预测
    并集成选出最终预测，输出预测标签 (0~11)"""
    # 使用概率累加的方式进行分类器集成
    probability = cls_list[0].predict_proba(data_inputs)
    for cls in cls_list[1:]:
        probability = np.add(probability, cls.predict_proba(data_inputs))
    # 取出最大的概率作为其标签
    predict = np.argmax(probability, axis=1)
    return predict


if __name__ == '__main__':
    # 获取所有特征
    train_features, train_labels, test_features = Step1_Get_Feature(is_preload)
    # K 折交叉验证
    ##############################################################################
    # 存交叉验证数据
    accuracy_list = []
    precision_list = []
    f1_list = []
    # K 折交叉验证
    kfold = KFold(n_splits=n_splits, shuffle=True, random_state=46)
    for train_index, val_index in kfold.split(train_features):
        print('================================================')
        train_X, val_X = train_features[train_index], train_features[val_index]
        train_Y, val_Y = train_labels[train_index], train_labels[val_index]
        # 训练好所有分类器
        cls_list = Step2_Train_Classification(train_X, train_Y)
        # 用验证集计算正确率等参数
        predict = Step3_Predict(val_X, cls_list)
        temp1 = accuracy(val_Y, predict)
        temp2, temp3 = evaluate(val_Y, predict)
        # 储存画图
        accuracy_list.append(temp1)
        precision_list.append(temp2)
        f1_list.append(temp3)
        # 使用模型进行预测
        predict = Step3_Predict(test_features, cls_list)
        Make_CSV(predict, f'submission_features{temp3}.csv')
```

```
    # 画图
    ##########################################################################
    import matplotlib.pyplot as plt

    plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
    plt.plot(range(1, n_splits + 1), accuracy_list, '-^', label="accuracy")
    plt.plot(range(1, n_splits + 1), precision_list, '-.o', label="precision")
    plt.plot(range(1, n_splits + 1), f1_list, ':.', label="L1 得分")

    plt.xticks(range(1, n_splits + 1))
    plt.ylim(0.5, 1)
    plt.xlabel('次数', loc='right')
    plt.ylabel('得分（0-1）', loc='top')
    plt.title("预测准确率等得分")
    plt.savefig(fname="预测准确率等得分.png", dpi=440)
    plt.legend()
    plt.show()
```

## 2. Image_Process.py

```
import cv2
import numpy
import os
from tqdm import tqdm

image_test = cv2.imread('data/train/Charlock/0edcd02cd.png')
# image_test = cv2.imread('data/train/Black-grass/0be707615.png')
# 绿色的HSV范围：色调-饱和度-明度
green_lower = numpy.array([35, 43, 46])  # 绿色下限为[35, 43, 46]
green_upper = numpy.array([77, 255, 255])  # 绿色上限为[77, 255, 255]
is_equalize = True
is_sharpening = False


def Image_Equalize(image):
    """将image彩色图像均衡化：对RGB通道分别均衡化后合并"""
    (B, G, R) = cv2.split(image)
    B = cv2.equalizeHist(B)
    G = cv2.equalizeHist(G)
    R = cv2.equalizeHist(R)
    # 合并每一个通道
    result = cv2.merge((B, G, R))
    return result


def Image_Sharpening(image):
    """对图像使用拉普拉斯算子进行锐化"""
    sharpen_op = numpy.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], 'float32')
    result = cv2.filter2D(image, cv2.CV_32F, sharpen_op)
    result = cv2.convertScaleAbs(result)
```

```
    return result


def Image_Split(mask, image, is_display=False):
    """将图像转换到 HSV 空间以筛选出图像中的绿色区域，其他区域置为 0"""
    # 先将 mask 模糊
    mask = cv2.GaussianBlur(mask, (3, 3), 0)
    # mask 转到 HSV 空间进行颜色分割
    mask = cv2.cvtColor(mask, cv2.COLOR_BGR2HSV)
    # 根据阈值找到对应颜色
    mask = cv2.inRange(mask, green_lower, green_upper)
    result = cv2.bitwise_and(image, image, mask=mask)
    # 展示图片
    if is_display:
        print(image.shape)
        cv2.imshow("images", numpy.hstack([image, result]))
        cv2.waitKey(0)
        exit()
    return result


def Process_All(file_from, file_to, number_limit, is_sharpening=True,
is_equalize=True, is_display=False):
    """处理并保存图片至新目录。
    file_from: 源根目录，"file_to+file_from": 目标目录，number_limit: 每个最底层文件夹下
文件个数（None 时为无限制）"""
    # 夹带私货……将后续要用到的 res 文件夹也先创建好
    if not os.path.exists('res'):
        os.makedirs('res')
    if not os.path.exists(file_from):
        print('未找到原始数据集，请完成以下 2 点操作以启动程序：\n'
              f'（1）请将原始数据集放置于当前路径下({sys.path[0]})\n'
              '（2）并将其根目录命名为 data')
        exit()

    for cur_dir, sub_dir, files in tqdm(os.walk(file_from)):
        # 通过 count 计数最底层文件夹下文件个数
        count = 0
        for file_name in files:
            if ((number_limit is None) or (count < number_limit)) and
file_name.endswith(('.png', '.jpg')):
                count += 1
                # 原路径: cur_dir + os.sep + file_name
                # 新路径: file_to + cur_dir + os.sep + file_name
                image = cv2.imread(cur_dir + os.sep + file_name)  # 从原地址处获取
                result = image
                # 进行图片处理
                # 均衡化
                if is_equalize:
                    result = Image_Equalize(result)
                # 锐化
                if is_sharpening:
```

```
                result = Image_Sharpening(result)
            # 去除背景
            result = Image_Split(image, result, is_display)

            # 若新目录不存在，则需要先创建
            temp file = file to + cur dir
            if not os.path.exists(temp_file):
                os.makedirs(temp_file)
            cv2.imwrite(temp file + os.sep + file name, result)  # 保存入新文件夹
下


def Process_All_StartUp():
    number_limit = input('你需要产生并保存一份处理后的图片数据吗？\n'
                         '  ------------------------------------\n'
                         '  *修改 Process_All()的 file_from、file_to 参数以更改路径*\n'
                         '  ------------------------------------\n'
                         '> 查看处理效果、或若不需要请输入 0\n'
                         '> 若需要限制每个文件夹内图片个数请输入一个正整数\n'
                         '> 若要处理并产生所有图片，请输入 None\n'
                         '> 请输入数量限制：')
    if number_limit == '0':
        Image_Split(image_test, image_test, is_display=True)
    else:
        if number_limit == 'None':
            number_limit = None
            file_to = 'all_'
        else:
            number_limit = int(number_limit)
            file_to = 'small_'
        Process_All(file_from='data', file_to=file_to, number_limit=number_limit,
                    is_sharpening=is_sharpening, is_equalize=is_equalize,
is_display=False)
        print('处理完成！')


if __name__ == '__main__':
    # Image_Split(image_test, image_test, True)
    # Process_All_StartUp()
    Process_All(file_from='data', file_to='all_', number_limit=None,
                is_sharpening=is_sharpening, is_equalize=is_equalize,
is_display=False)
```

### 3. Data_Load.py

```
import numpy
import torch.utils.data
import torchvision
from sklearn.model_selection import train_test_split

import os
```

```
from PIL import Image
from tqdm import tqdm

image_depth = 3
image_size = 256


def Data_Load(path):
    """读取给定路径下的 train、test 文件夹内所有图片
    返回三个 numpy，依次为：train 所有图片、train 所有标签、test 所有图片"""
    # Train
    print('train 文件夹文件读取中……')
    transforms = torchvision.transforms.Compose([
        # 缩放图像以创建 image_size x image_size 的新图像
        torchvision.transforms.Resize((image_size, image_size))])
    # 数据读取
    dataset = torchvision.datasets.ImageFolder(path + '/train', transforms)
    # 转化为 numpy
    train_dataset = numpy.array([numpy.array(image) for image, _ in
tqdm(dataset)])
    train_label = numpy.array([label for _, label in tqdm(dataset)])
    # Test
    print('test 文件夹文件读取中……')
    files = os.listdir(path + '/test')
    test_dataset = list()
    for i in tqdm(files):
        image = Image.open(path + '/test/' + i)
        image = image.resize((image_size, image_size), Image.ANTIALIAS)
        test_dataset.append(numpy.array(image))
    test_dataset = numpy.array(test_dataset)

    print(f'数据加载完成，train 图片数量为：{len(dataset)}')
    print(f'train_dataset: {train_dataset.shape},train_label:
{train_label.shape}\ntest_dataset: {test_dataset.shape}')
    return train_dataset, train_label, test_dataset


def Data_Load_CNN(path, validation_size=0.2):
    """从测试图片目录下读取所有文件，返回 CNN 网络的数据集 train_dataset、val_dataset
    path：源根目录，validation_size：验证集比例（默认 20%）
    dtype：返回 numpy（[n,size,size,depth]0~255 整数）或 tensor（[n,depth,
size,size]0~1 浮点数）"""
    transforms = torchvision.transforms.Compose([
        # 缩放图像以创建 image_size x image_size 的新图像
        torchvision.transforms.Resize((image_size, image_size)),
        # torchvision.transforms.RandomHorizontalFlip(),
        torchvision.transforms.ToTensor()
        # torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
    ])
    # Train
    print('train 文件夹文件读取中……')
```

```
    dataset = torchvision.datasets.ImageFolder(path + '/train', transforms)
    # 训练集、验证集划分
    val_size = int(validation_size * len(dataset))
    train_size = len(dataset) - val_size
    train_dataset, val_dataset = torch.utils.data.random_split(dataset,
[train size, val size])
    print(f'数据加载完成，train 图片数量为：{len(dataset)}')

    # Test
    files = os.listdir(path + '/test')
    test_dataset = list()
    for i in tqdm(files):
        image = transforms(Image.open(path + '/test/' + i))
        test_dataset.append(image)
    print(f'test 图片数量为：{len(test_dataset)}')
    return train_dataset, val_dataset, test_dataset


number2label = {0: 'Black-grass',
                1: 'Charlock',
                2: 'Cleavers',
                3: 'Common Chickweed',
                4: 'Common wheat',
                5: 'Fat Hen',
                6: 'Loose Silky-bent',
                7: 'Maize',
                8: 'Scentless Mayweed',
                9: 'Shepherds Purse',
                10: 'Small-flowered Cranesbill',
                11: 'Sugar beet'}

if __name__ == '__main__':
    # Data_Load_Train_Val('small_data/train', dtype='tensor')
    Data_Load('small_data')
```

## 4. Defines.py

```
import cv2
import numpy
import numpy as np
import pandas
import joblib  # 数据加载与保存
from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA  # PCA 降维

# XGBooster 分类器
from xgboost import XGBClassifier
# 支持向量机
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC, SVR
```

```
# 随机森林
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
# K 近邻
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

from Data_Load import *

path = 'all_data'  # 数据集位置
is_preload = False  # 是否读取预处理的数据

SIFT_BOW_size = 80  # Sift 中,词袋大小(降维后的特征数量)设置
HOG_PCA_size = 60  # HOG 降维后的特征数量设置
LBP_PCA_size = 80  # LBP 降维后的特征数量设置

n_splits = 5  # K 折交叉验证的 K 数


# 功能函数
############################################################
# 降维，将一个图片的 n 维特征，提取出 n 个重要特征，减少特征的数量
def pca(hist, n):
    print('\nPCA 降维中……')
    print(f'{hist.shape[1]}降至{n}维')
    pca = PCA(n_components=n)
    pca.fit(hist)
    hist_new = pca.transform(hist)  # 降维结果
    return hist_new


# 将多个特征，合并为一个特征
def merge(*wg):
    return np.hstack(wg)


# 正确率计算
def accuracy(actual, predict):
    # 计算正确的标签个数
    correct = (predict == actual).sum()
    total = len(actual)
    print(f'正确个数: {correct},总数: {total}, 正确率{correct / total:.6f}')
    return correct / total


def evaluate(actual, predict):
    """计算真实值和预测值间的精确率"""
    # 精确率 = 所有分类正确的正例/所有正例
    m_precision = metrics.precision_score(actual, predict, average="macro")
    print('m_precision: ', m_precision)
    # 召回率 = 提取出的正确信息条数 /样本中的信息条数
    m_recall = metrics.recall_score(actual, predict, average="macro")
```

```
    print('m_recall: ', m_recall)
    # F1 = 2 * (precision * recall) / (precision + recall)
    m_f1 = metrics.f1_score(actual, predict, average="macro")
    print('f1-score: ', m_f1)
    return m_precision, m_f1


# 生成 test 集的标签 csv 文档
def Make CSV(predict label, filename):
    """传入对 test 集图片的预测标签，生成并保存用于提交的 csv 文件"""
    # 将标签转换为植物名字
    predict_label = [number2label[i] for i in predict_label]
    # 保存到 csv 文件中
    files = os.listdir(path + '/test')
    file = pandas.DataFrame(columns=['file', 'species'], data=list(zip(files,
predict_label)))
    file.to_csv(path_or_buf=filename, index=False)


# 分类器生成
############################################################
def Clf_SVC(train_data, train_label):
    # SVC(核函数 kernel='linear'时变为为 SVM)
    clf = SVC(C=12, probability=True)  # default with 'rbf'
    print('选用分类器: ', clf)
    clf.fit(train_data, train_label)
    return clf


def Clf_SVR(train_data, train_label):
    # SVR
    clf = OneVsRestClassifier(SVR(C=12))
    print('选用分类器: ', clf)
    clf.fit(train_data, train_label)  # train_hist 为特征，根据特征和标签训练样本
    return clf


def Clf_RandomForestClassifier(train_data, train_label):
    # 随机森林分类器
    clf = RandomForestClassifier(n_estimators=400, max_features='sqrt',
max_depth=40,
                                 bootstrap=False, oob_score=False, random_state=10)
    # n_estimators（树棵数）、bootstrap（有放回采样，会有 37%左右不会被使用）
    print('选用分类器: ', clf)
    clf.fit(train_data, train_label)
    return clf


def Clf_ExtraTreesClassifier(train_data, train_label):
    # ExtraTree 分类器集合
    clf = ExtraTreesClassifier(n_estimators=400, max_features='sqrt',
max_depth=50)
```

```
    print('选用分类器：', clf)
    clf.fit(train_data, train_label)
    return clf


def Cls_XGBooster(train_data, train_label):
    # xgboost
    clf = XGBClassifier(learning_rate=0.13, max_depth=5, n_estimators=300,
nthread=10,
                        use_label_encoder=False, eval_metric='mlogloss')
    print('选用分类器：', clf)
    clf.fit(train_data, train_label)
    return clf


def Clf_KNN(train_data, train_label):
    # KNN
    clf = KNeighborsClassifier()
    print('选用分类器：', clf)
    clf.fit(train_data, train_label)
    return clf
```

## 5. SIFT.py

```
from scipy import cluster

from Defines import *


def Sift(dataset):
    """提取 sift 特征，每张图片特征维度为[x, 128]"""
    # List where all the descriptors are stored
    sift_list = []
    fail_count = 0
    print('1、开始进行所有图片的 sift 特征运算')
    detector = cv2.xfeatures2d.SIFT_create()
    for image in tqdm(dataset):
        # des:[x, 128]
        kp, sift = detector.detectAndCompute(image, None)  # des 是描述子 sift 是关键
点特征，每张图关键点不一样多，每个关键点是 128 维
        if sift is None:
            fail_count += 1
            sift = sift_list[len(sift_list) - 1]
        sift_list.append(sift)
    print(f' *发现{fail_count}张图片未计算出 sift 特征*')
    return sift_list


def Kmeans(sift_list, WOB_size):
    """将所有图片特征进行 K 聚类 -> WOB_size 类，返回所有聚类中心"""
    # 将所有特征纵向堆叠起来,每行当做一个特征词 n*[x, 128] -> [ni*xi, 128]
```

```
    print('2.1、开始堆叠所有 sift 特征')
    descriptors = numpy.vstack(sift_list)
    print(f'shape 变化：({len(sift_list)}, x, 128) -> {descriptors.shape}')
    # 对所有特征进行 K 聚类，共 WOB_size 类
    print("2.2、开始 K 聚类：聚类中心个数%d，数据点个数%d （这个过程耗时最长）" %
(WOB size, descriptors.shape[0]))
    # kmeans 函数输出分别为：聚类中心、损失 distortion
    cluster_dict, _ = cluster.vq.kmeans(descriptors, WOB_size, iter=1)
    return cluster dict


def BOW(sift_list, BOW_size, cluster_dict):
    """
    统计各图片特征的相对词袋的 idf（词的频繁度），并归一化
    sift_list: 各图片 sift 特征列表
    WOB_size: 词袋（字典）大小
    cluster_dict: 词袋（字典）中所有单词
    """
    # 词袋模型 [单词文档共现矩阵 TF-IDF]
    # im_features 为单词文档共现矩阵，每行表示一副图像，每列表示一个视觉词，统计每副图像中视觉
词的个数
    # [n, BOW_size]
    im_features = numpy.zeros((len(sift_list), BOW_size), "float32")
    for i in range(len(sift_list)):
        # 计算每副图片的所有特征向量和 voc 中每个特征 word 的距离，返回为匹配上的 word
        # 根据聚类中心将所有数据进行分类：sift_list[i][1]为数据，voc 是 kmeans 产生的聚类中
心
        # vq 输出：数据属于哪一中心,与各中心的损失
        words, _ = cluster.vq.vq(sift_list[i], cluster_dict)
        for w in words:
            im_features[i][w] += 1

    # TFw：单词 w 出现在 TFw 个文档中
    TFw = numpy.sum((im_features > 0) * 1, axis=0)
    # IDFw = log(文档总数 N / TFw)，log(x+1)防止 x 为 0 导致数值错误
    # [BOW_size,]
    IDFw = numpy.array(numpy.log((len(sift_list) + 1) / (TFw + 1)), 'float32')
    # [n, BOW_size]
    im_features = im_features * IDFw
    # L2 归一化
    im_features = preprocessing.normalize(im_features, norm='l2')
    return im_features


# 获得特征并保存
############################################################
def Train_Sift(train_data, train_label, test_data):
    # 将 train 和 test 集合并到一起进行特征提取（train, test）
    all_image = numpy.concatenate((train_data, test_data))
    sift_list = Sift(all_image)  # 提取 sift 特征
    cluster_dict = Kmeans(sift_list, SIFT_BOW_size)  # K 聚类获得词典
    all_features = BOW(sift_list, SIFT_BOW_size, cluster_dict)  # 通过词袋模型计算图
```

```
片所含各单词频率数据
    # 保存时，将 train 和 test 集合分开
    joblib.dump((all_features[:len(train_data)], train_label,
all_features[len(train_data):]),
                f'res/sift_features_bow={SIFT_BOW_size}.pkl')


if __name__ == '__main__':
    # is preload = False
    if not is_preload:
        train_data, train_label, test_data = Data_Load(path)
        Train_Sift(train_data, train_label, test_data)
    # 使用 Sift 特征进行预测
    train_data, train_label, test_data =
joblib.load(f'res/sift_features_bow={SIFT_BOW_size}.pkl')
    train_X, val_X, train_Y, val_Y = train_test_split(train_data, train_label,
test_size=0.2, random_state=1)
    # 训练分类器、进行预测、计算正确率
    clf = Clf_SVC(train_X, train_Y)
    pred = clf.predict(val_X)
    accuracy(val_Y, pred)
```

## 6. HOG.py

```
from skimage import feature as ft
from Defines import *


# HOG 特征提取
############################################################
def HOG_detect(image_dataset):
    """输入所有图片
    计算所有图片的 HOG 特征（纹理特征）
    并降维后返回"""
    temp = ft.hog(cv2.cvtColor(image_dataset[0], cv2.COLOR_RGB2GRAY),
                  orientations=6, pixels_per_cell=[8, 8], cells_per_block=[3, 3])
    hist = np.zeros((len(image_dataset), temp.shape[0]), "float32")
    print('\n 下面开始 HOG 特征提取')
    for i in tqdm(range(len(image_dataset))):
        gray = cv2.cvtColor(image dataset[i], cv2.COLOR RGB2GRAY)
        hist[i] = ft.hog(gray, orientations=6, pixels_per_cell=[8, 8],
cells_per_block=[3, 3])
    return pca(hist, HOG_PCA_size)


# 获得特征并保存
############################################################
# 计算 HOG 特征并保存
def Train_HOG(train_data, train_label, test_data):
    # 将 train 和 test 集合并到一起进行特征提取（train, test）
    all_image = numpy.concatenate((train_data, test_data))
```

```
    all_features = HOG_detect(all_image)
    # 保存时，将 train 和 test 集合分开
    joblib.dump((all_features[:len(train_data)], train_label,
all_features[len(train_data):]),
                f'res/hog_features_pca={HOG_PCA_size}.pkl')


if __name__ == '__main__':
    # is preload = False
    if not is_preload:
        train_data, train_label, test_data = Data_Load(path)
        Train_HOG(train_data, train_label, test_data)

    # 使用 HOG 特征进行预测
    train_data, train_label, test_data =
joblib.load(f'res/hog_features_pca={HOG_PCA_size}.pkl')
    train_X, val_X, train_Y, val_Y = train_test_split(train_data, train_label,
test_size=0.2, random_state=1)
    # 训练分类器、进行预测、计算正确率
    clf = Clf_SVC(train_X, train_Y)
    pred = clf.predict(val_X)
    accuracy(val_Y, pred)
```

## 7. LBP.py

```
from skimage import feature as ft
from Defines import *

# LBP 特征提取
############################################################
def LBP_detect(image_dataset):
    """输入所有图片
    计算所有图片的特征
    并降维后返回"""
    radius = 1
    n_point = radius * 8
    hist = np.zeros((len(image_dataset), image_size), "float32")
    print('\n 下面开始 LBP 特征提取')
    for i in tqdm(range(len(image_dataset))):
        gray = cv2.cvtColor(image dataset[i], cv2.COLOR RGB2GRAY)
        lbp = ft.local_binary_pattern(gray, n_point, radius, 'default')
        max_bins = int(lbp.max() + 1)
        hist[i], _ = np.histogram(lbp, density=True, bins=max_bins, range=(0,
max_bins))
    return pca(hist, LBP_PCA_size)

# 获得特征并保存
############################################################

# 计算 LBP 特征并保存
def Train_LBP(train_data, train_label, test_data):
```

```
    # 将 train 和 test 集合并到一起进行特征提取（train, test）
    all_image = numpy.concatenate((train_data, test_data))
    all_features = LBP_detect(all_image)
    # 保存时，将 train 和 test 集合分开
    joblib.dump((all_features[:len(train_data)], train_label,
all features[len(train data):]),
                f'res/lbp_features_pca={LBP_PCA_size}.pkl')

if   name   == '  main  ':
    # is_preload = False
    if not is_preload:
        train_data, train_label, test_data = Data_Load(path)
        Train_LBP(train_data, train_label, test_data)

    # 使用 LBP 特征进行预测
    train_data, train_label, test_data =
joblib.load(f'res/lbp_features_pca={LBP_PCA_size}.pkl')
    train_X, val_X, train_Y, val_Y = train_test_split(train_data, train_label,
test_size=0.2, random_state=1)
    # 训练分类器、进行预测、计算正确率
    clf = Clf_SVC(train_X, train_Y)
    pred = clf.predict(val_X)
    accuracy(val_Y, pred)
```