摘要

植物的种类繁多, 对植物进行科学的分类至关重要。叶片的识别与分类是识别植物的重要组成部分。本文基于 100 种植物树叶的二值化图片数据, 通过**解析几何计算**和**时间序列展开**的方法转换成**轮廓特征向量**和**边缘特征向量**; 再通过**深度神经网络 (DNN)** 解决树叶识别分类问题, 引入**粒子群 (PSO)** 算法优化网络中的连接权值与阈值, 并结合树叶纹理数据信息对模型进行改进和分析比较。

针对问题一，通过**解析几何计算**和**时间序列展开**，分别提取每一张图片中的特征向量。根据题目要求，本组首先利用 matlab 计算与图像形状相关的解析几何特征量，得到由八个几何学、拓扑学特征值组成的**八维**形状特征向量。然后将图像轮廓进行极化投影，并通过 numpy 工具箱将极化投影轮廓展开成时间序列，挖掘分析时间序列的三个特征值，组成**三维**边缘特征向量。最后合并两个向量得到**十一维**总体特征向量。

针对问题二, 基于问题一中提取的 11 维度的特征信息, 采用 **Keras 工具库**搭建**深度神经网络 (DNN)**, 利用**粒子群 (PSO)** 算法优化 DNN 网络的连接权值和阈值, 建立了**PSO-DNN 模型**对树叶进行识别与分类。采用网格搜索法确定了参数的选取, 训练后的神经网络模型预测准确率为 **91.237%**, 并给出各指标的权重占比。相较于其他神经网络如 Elman、BP 和 DNN 动态网络,PSO-DNN 网络的准确率最高, 收敛性最强。

针对问题三, 基于问题二中的 **PSO-DNN 网络模型**, 并结合树叶纹理数据信息, 增加网络输入层个数, 对模型进行参数调整, 最终叶片的预测准确率达到 **96.634%**

本文中所提到的模型优点主要有两点：一、选取的指标从多维度、全方面考虑，收集的数据真实可靠；二、利用灰色关联度筛选出三类人才的人才吸引力要素，相较于人工选取指标具有客观性、严谨性。

关键词： 时间序列　深度神经网络　粒子群算法　网格搜索法

目录

一、 问题重述

1.1 问题背景

植物的种类繁多, 要了解和掌握如此多的植物, 必须进行一个科学的分类。人们常常根据植物的用途，或根据植物的一个或几个明显的形态进行分类，植物的识别与分类对于区分植物种类, 探索植物间的亲缘关系, 阐明植物系统的进化规律具有重要的意义。因此植物分类学是植物科学甚至整个生命科学的基础学科。目前对于树叶识别与分类主要由人完成, 但是树叶种类庞大, 依赖人工地进行树叶识别与分类是不现实的。所以树叶的研究对于植物总体的研究能提供很大的帮助。

从树叶的各个方面, 纹理, 硬度, 离心率等方面都可作为主要方向研究, 现对树叶的研究主要通过采集树叶图形, 利用数字图像处理来对树叶进行分类识别, 这种方法只停留在处理形态特征, 有很大的局限性, 忽略了树叶的生理特征和其他特征, 所以研究方法来综合处理树叶平面图像特征, 形态特征和生理特征很有必要性。

1.2 问题提出

围绕植物分类进行树叶识别与分类, 以树叶二值化图片为依据, 依次提出以下问题:

(1) 结合附件中的二值化的图片数据, 建立合适的图片数据提取方案, 量化处理图片数据, 并具体分析说明所提取数据信息的量化指标体系。

(2) 基于问题一中提取的数据信息, 建立合适的数学模型由数据出发判断叶子的种类, 研究判别分类的核心指标, 并估计出模型的性能以及核心指标对模型判别性能的影响。

(3) 基于二值化图片数据, 结合附件中叶子纹理的数据信息, 对原有模型进行改进, 并对新旧模型进行比较分析。

二、 模型假设

(1) 不考虑叶片的枝干和叶柄的数据特征

(2) 假设二值化的树叶叶片数据没有残损卷曲。

(3) 为保证分类方法具有可推广性，假设现实中同种类树叶与题目所给的树叶二值化图像具有相似的形状轮廓。

三、 符号说明

符号	说明
I	研究图像
$A(I)$	图像面积
$C(I)$	图像几何中心
∂I	图像边界
$d(.)$	运算两点间欧式距离
ID	总特征向量
ID_{shape}	形状特征向量
ID_{margin}	边缘特征向量
id_k	研究图像特征值
r	极径
θ	极角
X	时间序列集
S	shapelet 子序列
x_i	神经网络第 i 个输入值
w_{ki}	第 i 个输入量连接的权值
b_k	神经网络阈值
f	激活函数
e	误差函数
E	全局误差
η	学习率
$c_{1,2}$	加速因子
β	惯性权重

四、 问题一模型的建立与求解

4.1 问题的描述与分析

针对问题一，本题要求建立合适方案提取二值化图片中的数据，并对所提取数据的量化指标进行分析说明。本组通过解析几何计算和时间序列展开，将目标图像转化成两个特征向量。本组根据题目要求和近年的图像识别研究，确定了针对二值化图像的两个重点识别因素——形状特征因素和边缘特征因素。针对形状特征因素，首先利用 matlab 做解析几何运算，计算与图像轮廓有关的特征量，并将计算结果作为元素，组成轮廓特征向量。针对边缘特征因素，首先将图像边缘通过极化投影展开为时间序列，计算每支时间序列的特征量，并将计算结果作为元素，组成边缘特征向量。最后合并两个向量得到总体特征向量。其具体特征值如图 1 所示：

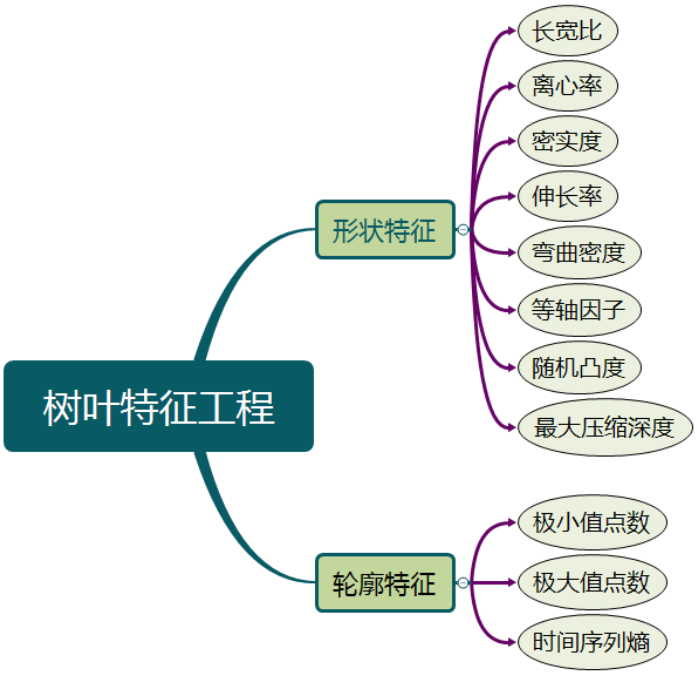


图 **1**　图形特征示意图

4.2 模型的建立

4.2.1 图像基本参数与运算符号定义

定义 I 表示目标树叶所对应的图像，∂I 表示图像边界，$D(I)$ 表示图像最小外接圆直径，$d(I)$ 表示图像最大内切圆半径，$A(I)$ 表示研究对象面积,$L(\partial I)$ 表示研究对象的轮廓线长度，$H(I)$ 表示研究对象的凸包域，$C(I)$ 为图像几何中心坐标，运算符 $d(.)$ 代表欧式距离。

4.2.2 形状特征因素

定义轮廓特征向量为 $ID_{shape} = [id_1, id_2, \cdots, id_n](n = 6)$ 其中 $id_k(k = 1, 2, \cdots, 6)$ 为二值化矩阵的轮廓特征，其具体计算公式如下：

(1) 长宽比 (Aspect Ratio): 定义 X_I 为图像最上方非零行与最下方非零行的行数差 (长)，Y_I 为图像最左方非零列与最右方非零列的列数差 (宽)，即长宽比计算公式：

$$id_1 = X_I / Y_I \tag{1}$$

(2) 离心率 (Eccentricity): 定义 $E(I)$ 是与研究图像具有相同的二阶矩的椭圆,a 和 b 分别为 $E(I)$ 对应的长轴与短轴，即离心率计算公式：

$$id_2 = \sqrt{1 - (\frac{b}{a})^2} \tag{2}$$

(3) 密实度 (Solidity): 反映研究对象的仿射特征的变量, 即研究对象区域的固靠性程度，计算公式：

$$id_3 = \frac{A}{A(H(I))} \tag{3}$$

(4) 等周因子 (Isoperimetric Factor): 描述目标树叶轮廓规整度的变量，变化范围为 (0,1)，叶子边缘越规则, 其值越接近于最大值 1，计算公式：

$$id_4 = \frac{4\pi \cdot A}{L(\partial I)^2} \tag{4}$$

(5) 伸长率 (Elongation): 描述研究对象向某方向伸展趋势的变量, 变化范围为 (0,1), 树叶越趋于圆形相应的伸长率越小，计算公式：

$$id_5 = 1 - \frac{2d_I}{D(I)} \tag{5}$$

□

(6) 最大压痕深度 (MaximalIndentationDepth): 定义 $C_{H(I)}$ 为研究对象凸型区域的几何中心,$L(\partial I)$ 表示为 $H(I)$ 的轮廓线长度,$\forall X \in H(I)$ 和 $\forall Y \in \partial I$, 计算距离 $d(X, C_{H(I)})$ 和 $d(Y, C_{H(I)})$。即最大压痕深度计算公式为：

$$max\left\{\frac{d(X, C_{H(I)}) - d(Y, C_{H(I)})}{L(H(I))}\right\} \tag{6}$$

(7) 随机凸性 (Stochastic Convexity): 记随机给定两个端点 $P_1, P2 \in \partial I$，记 $P(G)$ 为线段 $[XY]$ 完全包含于图像区域 I 中的概率，即随机凸性计算公式为：

$$id_7 = P(G) \tag{7}$$

(8) 弯曲能量 (Bending Energy): 描述研究对象边界弯曲程度的值，定义 φ_n 为 ∂I 的曲率，即弯曲能量计算公式为：

$$id_8 = 1/P \sum_{i=0}^{n-1} |\varphi_n - \varphi_{n-1}| \tag{8}$$

4.2.3 边缘特征因素

为得到研究对象的边缘特征，首先以 $C(I)$ 为坐标原点建立笛卡尔坐标系，对于曲线 ∂I 上任意一点 P 可以在该坐标系下表示为 $P(x_p, y_p)$，将其投影至以 $C(I)$ 为极点的极坐标系得 $P'(r_p, \theta_p)$，其中：

$$r_p = d(P(x_p, y_p).C(I)) \tag{9}$$

$$\theta_p = y_p / x_p, \tag{10}$$

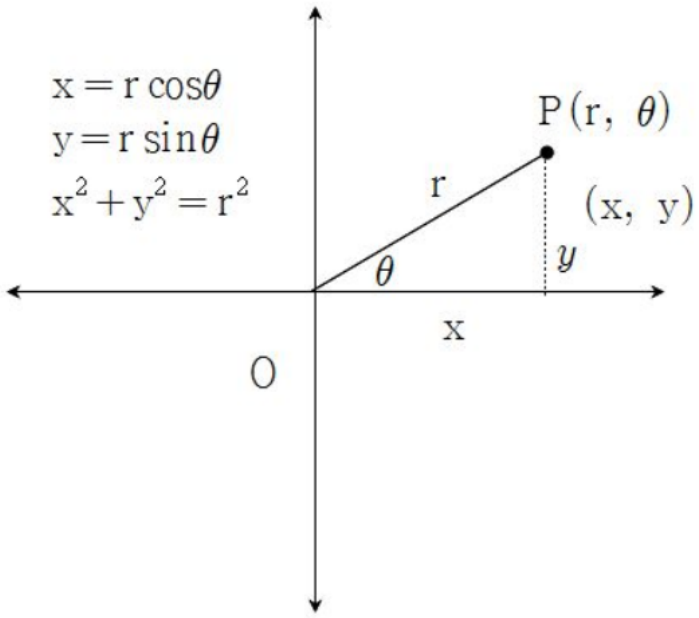


图 **2**　极化投影实意图

定义点集 $P'(r_p, \theta_p)$ 为研究样本，其中中包含 d' 个一元时间序列，Q 为单个样本一维的长度。将数据集中每个样本第 j 维 $(j \in \{1, 2, \cdots, d\})$ 数据组成一个一元时间序列数据集，记为 X。且其中表类样本 $y_i \in \{1, 2, \cdots, C\}, i \in \{1, \cdots, N\}$。

得到一元时间序列集 X 后，记 N 为序列集中的时间序列条数。在数据集 D 中有 n_i 个实例，并且 $n_1 + n_2, \cdots, n_n = N$ 计算 X 序列集熵值作为边缘特征值：

$$id_9 = -\sum_{i=1}^{C} \frac{n_i}{N} \log \frac{n_i}{N} \tag{11}$$

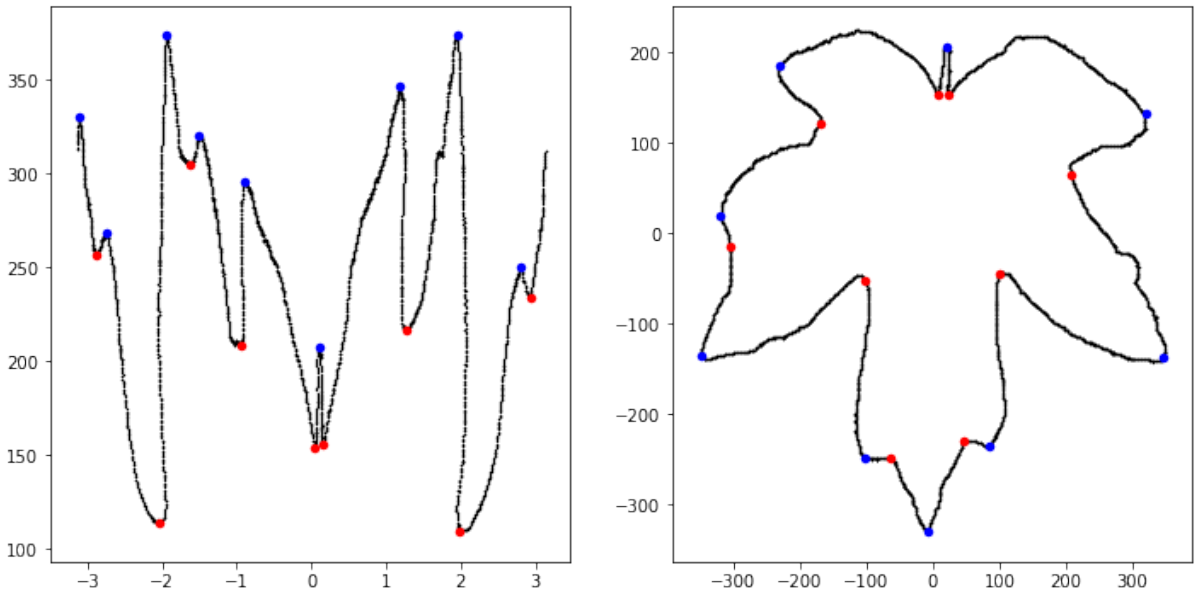
定义 $shapelet$ 为时间序列集 X 中能够最大程度区分不同类别时间序列的子序列。在一元时间序列分类问题中选出 K 个最优 shapelet 并记为 $S \in R^{K \times M}$(其中 M 为 shapelet 的长度。一个长度为 M 的 shapelet

$$S_k = \{s_{k1}, s_{k2}, \cdots, s_{kn}\} (k \in \{1, \cdots, K\}) \tag{12}$$

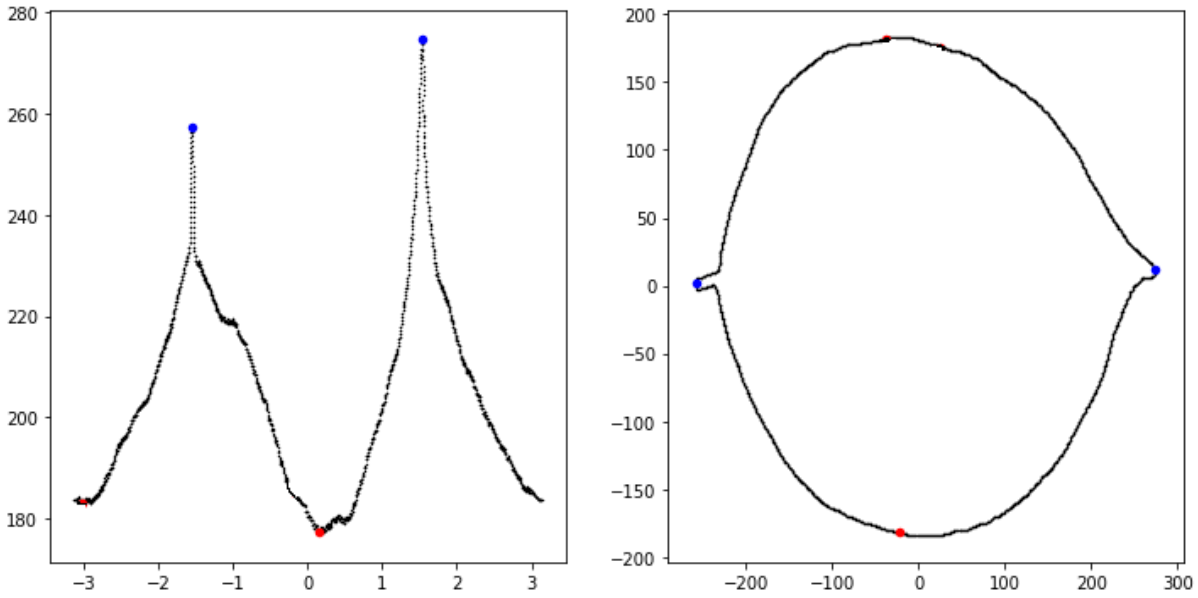
是时间序列 $X_i(i \in \{1,2,\cdots,d\})$ 的子序列，定义 X_i 与 S_k 之间的距离为：

$$D_{i,k} = \min_{j=1,\cdots,j} \frac{1}{M} \sum_{m=1}^{M} (X_{i,j+m-1} - S_{k,m})^2 \tag{13}$$

其中 M 表示所选定的 X_i 与 S_k 的子序列长度。再找到在找到 K 个最优 shapelet 后，计算 K 个最优 shapelet 与一个时间序列间的距离作为该时间序列的新特征，时间序列样本集被映射至新特征空间，最终将之间序列降至一维时间序列 arrays，展开效果如图 3 所示：



(a) Acer-Campestre



(b) Cornus-Controversa

图 **3**　时间序列展开效果图

将 arrays 滤波处理后，计算每支时间序列上的极大值点数 id_{10}，极小值点数 id_{11}。得到边缘特征向量 $ID_{margin} = [id_9, id_{10}, id_{11}]$，合并轮廓特征向量与边缘特征向量得：

$$ID = [ID_{shape}, ID_{margin}] \tag{14}$$

其中 ID 为总体特征向量。

4.3 模型的求解

首先使用 matlab 围绕 regionprops 函数对研究对象进行解析几何计算，用遍历式算法逐一算出研究对象长宽比、离心率、实密度、等周因子、伸长率和最大压痕深度，得到轮廓特征向量。再将图片通过 numpy 工具箱标准正立化，以其几何中心为极点将其边缘坐标转换为极坐标，然后使用 ndarrays 函数将所得极坐标降维展开成时间序列，搜寻滤波后时间序列的极点数得到边缘特征向量。其算法流程图如图 4

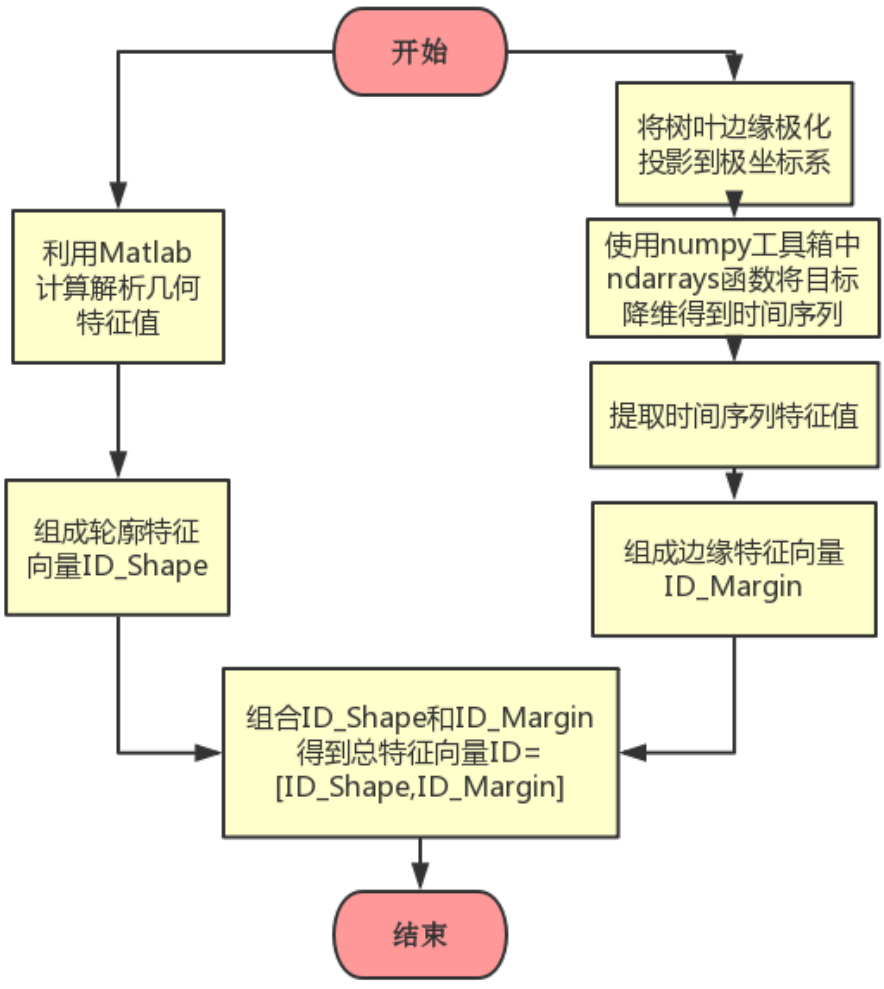


图 **4**　树叶特征工程示意图

五、 问题二模型的建立与求解

5.1 问题的描述与分析

问题二要求针对所提取数据信息，建立数学模型判别叶子的种类并研究其核心指标。本组通过问题一中所提取的 11 维度的特征，采用 $Keras$ 工具库搭建深度神经网络（Deep Neural Network）解决树叶多分类问题，并输出各维度特征的权重作为指标贡献率。DNN 神经网络采用 BP 算法基于梯度信息来调整连接权值，因而初始权值和阈值选取的随机性会导致网络稳定性差。为克服此缺点，引入收敛速度快、全局搜索能力强的粒子群 (PSO) 算法来优化 DNN 网络的连接权值和阈值，建立一种新的 $PSO-DNN$ 网络模型实现对树叶分类。当考虑到步长等参数的选取时，使用网格搜索的方式对参数进行优化，其流程图如下：

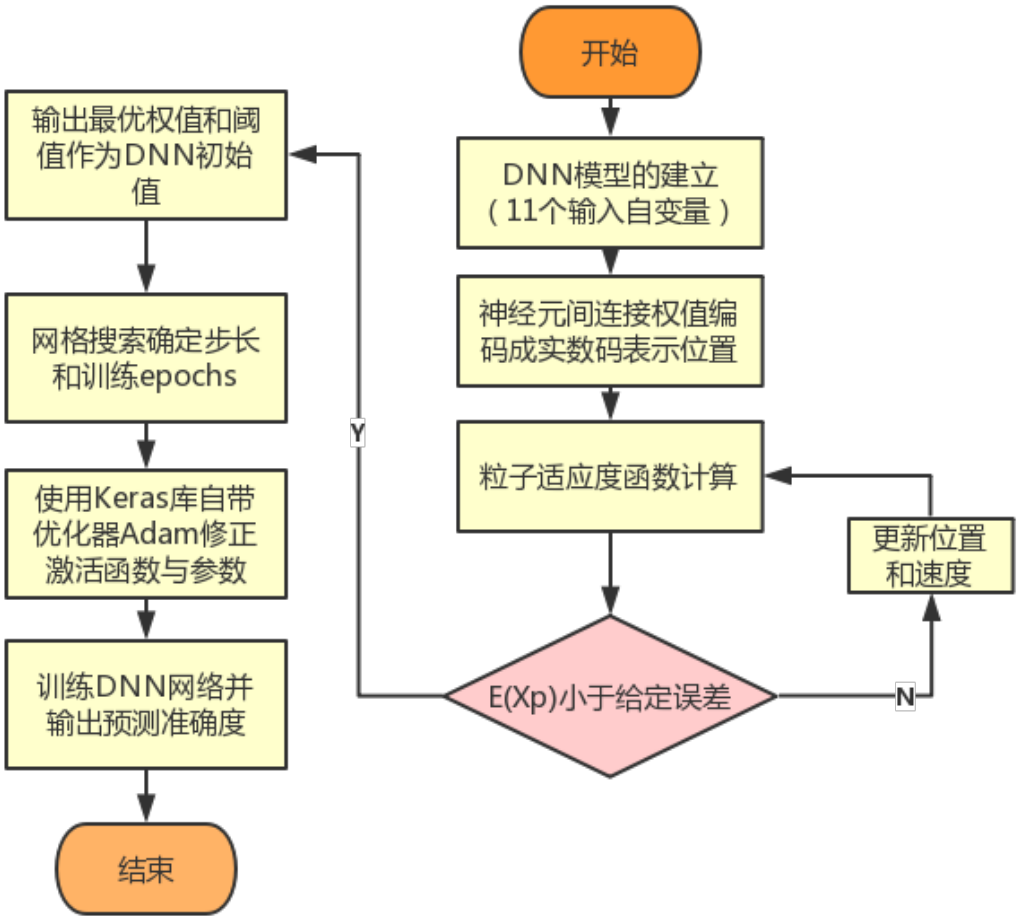


图 **5**　**PSO** 算法优化 **DNN** 网络流程图

5.2 模型的建立与求解

5.2.1 DNN 网络搭建

DNN 的正向传播主要依靠众多神经元的计算来完成的，此处设计的网络结构如图 6 所示，工作过程中可以用下式来表示：

$$u_k = \sum_{i=1} w_{ki} x_i \tag{15}$$

$$y_k = f\left(u_k - b_k\right) \tag{16}$$

其中:x_i 表示第 i 个输入; w_{ki} 表示与第 i 输入量相连的权值; u_k 表示所有输入的加权和; b_k 为神经元阈值;f 为激活函数; y_k 为神经网络的输出。

激活函数的种类有很多，如 $sigmoid$，$tanh$ 及 $Relu$，本文应用的是 $Relu$ 作为激活函数搭建两层隐含层，如式 17 所示：

$$\begin{aligned} f_{\text{Relu}} &= \max(0, z) \\ \frac{d}{dz} f_{ReLU} &= \begin{cases} 1, & z > 0 \\ 0, & z \leqslant 0 \end{cases} \end{aligned} \tag{17}$$

网络的输出层使用 $softmax$ 函数作分类器，式 18 为第 i 个神经的输出：

$$f_{\text{softinax}} = \mathrm{e}^i / \sum_j \mathrm{e}^j \tag{18}$$

具体算法步骤如下：

(1) 进行网络的初始化设置，用 $(-1, 1)$ 之间的随机数对 DNN 各个层的权值和阈值分别进行相应初始化，设定误差函数为 e、计算精度为 ϵ、学习速率为 η 以及最大学习次数为 M。

(2) 随机选取第 k 次输入样本 $x(k)$ 以及对应的实际期望输出 $d(k)$ 如下:

$$x(k) = [x_1(k), x_2(k), ..., x_n(k)]$$
$$d(k) = [d_1(k), d_2(k), ..., d_r(k)]$$

(3) 计算 DNN 中第 1 隐含层、第 2 隐含层以及输出层的输入值和输出值具体计算公式如下:

$$h_{\text{bs}}(k)=\sum_{i=1}^{n}u_{is}x_i(k)+b_s\quad s=1,2,\cdots,p$$

$$h_{\text{os}}s(k)=f_1\left(h_{\text{bs}}(k)\right)\quad s=1,2,\cdots,p$$

$$g_{\text{bt}}(k)=\sum_{s=1}^{p}v_{st}h_{\text{os}}(k)+b_t\quad t=1,2,\cdots,q$$

$$g_{\text{ot}}(k)=f_2\left(g_{\text{bt}}(k)\right)\quad s=1,2,\cdots,q$$

$$y_{\text{bj}}(k)=\sum_{i=1}^{q}w_{ij}g_{\text{at}}(k)+b_j\quad j=1,2,\cdots,r$$

$$y_{\text{oj}}(k)=f_3\left(y_{bj}(k)\right)\quad j=1,2,\cdots,r$$

其中 x_i 输入层输入向量,h_{b} 为第 1 隐含层输入向量,h_{o} 为第 1 隐含层输出向量，g_{b} 为第 2 隐含层输入向量,g_{o} 为第 2 隐含层输出向量，y_{b} 为输出层输入向量，y_{o} 为输出层输出向量输入向量

(4) 首先给定误差函数如下：

$$e=\frac{1}{2}\sum_{j=1}^{r}\left(d_j(k)-y_{bj}(k)\right)^2$$

误差函数 e 对输出各层求偏导数得：

$$\frac{\partial e}{\partial y_{bj}(k)}=\delta_j(k)$$
$$\frac{\partial e}{\partial g_{bt}(k)}=\delta_t(k)$$
$$\frac{\partial e}{\partial h_{bs}}(k)=\delta_s(k)$$

即通过 δ 值与各层的输出修正各层级间的权值, 其计算公式如下：

$$w_{tj}^{N+1}=w_{tj}^{N}+\eta\delta_j(k)g_{ot}(k)$$
$$w_{st}^{N+1}=w_{st}^{N}+\eta\delta_t(k)g_{os}(k)$$
$$w_{is}^{N+1}=w_{is}^{N}+\eta\delta_j(k)g_{ot}(k)$$

同理，各层级阈值变化公式如下

$$\Delta b_j(k)=\eta\delta_j(k)$$
$$\Delta b_l(k)=\eta\delta_t(k)$$
$$\Delta b_s(k)=\eta\delta_s(k)$$

使用当前权值计算全局误差，公式如下：

$$E=\frac{1}{2m}\sum_{k=1}^{m}\sum_{j=1}^{r}\left(d_j(k)-y_{oj}(k)\right)^2$$

当误差小于预设精度 ε 或达到最大学习次数 M 时网络算法结束, 否则返回 (2) 进行下一轮学习训练.

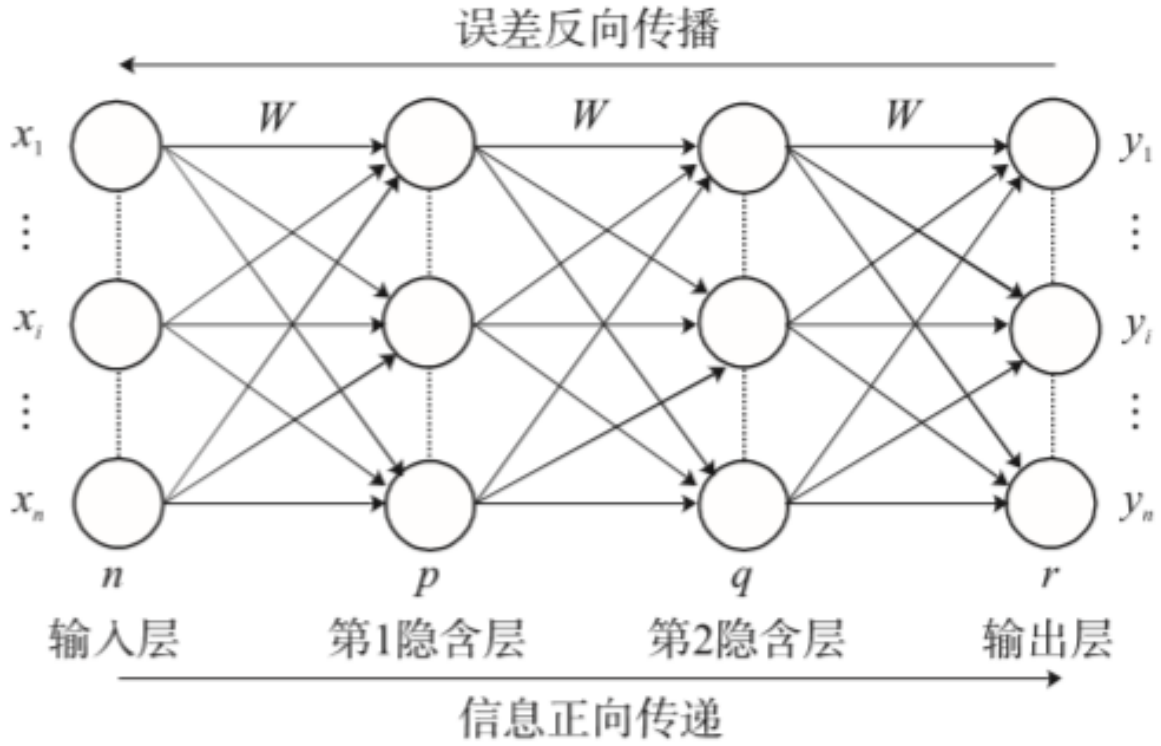


图 **6**　人工搭建 **DNN** 结构示意图

5.2.2 PSO 优化 DNN 网络

算法的具体实现步骤如下：

(1) 将 DNN 网络结构中所有神经元间的连接权值和各个神经元的阈值编码成实数码串表示的个体作为 PSO 算法要寻优的位置向量。

(2) 在编码空间中随机生成一定数目的个体组成种群，其不同个体代表神经网络不同权值。

(3) DNN 网络训练及个体的适应度评价。将微粒群中的每一个个体的分量映射为网络中的权值和阈值，从而构成个体对应的神经网络。首先划分训练样本和测试样本; 其次输入训练样本进行网络训练，通过反复迭代来优化网络权值，并计算每一个网络在训练集上产生的均方误差,，以此作为目标函数; 最后对每个个体进行适应度评价，从中找到最佳个体用来判断是否需要更新微粒的 Gbest 与 Pbest，以全局误差值作为适应度函数:

$$E = \frac{1}{2m}\sum_{k=1}^{m}\sum_{j=1}^{r}(d_j(k) - y_{oj}(k))^2$$

式中：m 为训练样本个数，r 为输出端个数，$y_{oj}(k)$ 为训练样本在输出端的给定输出，$d_j(k)$ 为训练样本在输出端的实测值，他们俩个值的误差平方和越小，表示实际值和预测值越接近，网络的性能越好。

(4) 更新每个粒子的速度和位置，产生下一代的粒子群。更新公式如下:

$$v_{id}^{t+1} = v_{id}^{t} + c_1 r_1 (p_{id}^{t} - x_{id}^{t}) + c_2 r_2 (g_{id}^{t} - x_{id}^{t}) \tag{19}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{20}$$

其中 $i = 1, 2, ..., n; d = 1, 2, ..., d; t$ 为当前迭代次数；v_{id}^{t} 为当前粒子速度 (t 时刻)；x_{id}^{t} 为当前粒子位置 (t 时刻)；$(p_{id}^{t} - x_{id}^{t})$ 为当前位置与自己最好位置之间的距离；$(g_{id}^{t} - x_{id}^{t})$ 为当前位置与群体最好位置之间的距离；v_{id}^{t+1} 为下一时刻粒子速度 ($t+1$ 时刻)；c_1, c_2 为非负常数，称为加速因子；r_1, r_2 为均匀分布与 $[0, 1]$ 区间的随机数。

(5) 当目标函数小于给定的误差或达到最大迭代次数时，算法结束。将 PSO 算法训练出来的最佳神经网络的权值和阈值作为 DNN 网络的初始值，并记录。

5.2.3 参数的选取与确定

由于训练 epochs 与步长选取较为困难，实验采取**网格搜索法**（Grid Search）来确定参数的选取，网格搜索是指定参数值的一种穷举搜索方法，通过将估计函数的参数通过交叉验证的方法进行优化来得到最优的学习算法。

在迭代修正的过程中，实验采用 $Keras$ 自带优化器 **Adam 函数**进行迭代修正，如式 21 所示，其中 β_1 一般取 0.9，β_2 一般取 0.999。

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w f(w_t) \\ v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla_w f(w_t)^2 \\ \widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\ w_t = w_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} \end{cases} \tag{21}$$

5.3 实验与结果分析

随机选取 80% 的数据作为训练集进行数据**标准化处理**，再对树叶种类（species）进行 **One-Hot 编码**作为分类目标。训练输入样本为 $1280 * 11$，测试输入样本为 $320 * 11$，DNN 网络输入层神经元 11 个，输出层神经元 100 个，包含 2 个隐含层，适应度函数选择均方误差。按照 PSO 优化 DNN 网络模型的步骤不断地迭代寻找最优网络参数，进行树叶分类预测的仿真实验。具体参数设置间表 1 。

表 **1**　参数设置表

参数名称	参数符号	参数值
选择粒子个数为	T	30
计算精度	ε	0.00001
学习率	η	0.01
最大迭代次数	M	500
维度	S	11
加速因子 1	c_1	1.49
加速因子 2	c_2	1.49
惯性权重	β	0.9

通过上述参数设置，训练后各维度权重占比如图 8 所示，最小均方误差为 0.458，**预测准确率为** 91.037%。其迭代过程中损失与准确率如下图所示：

图 **7**　**PSO-DNN** 迭代过程中损失与准确率

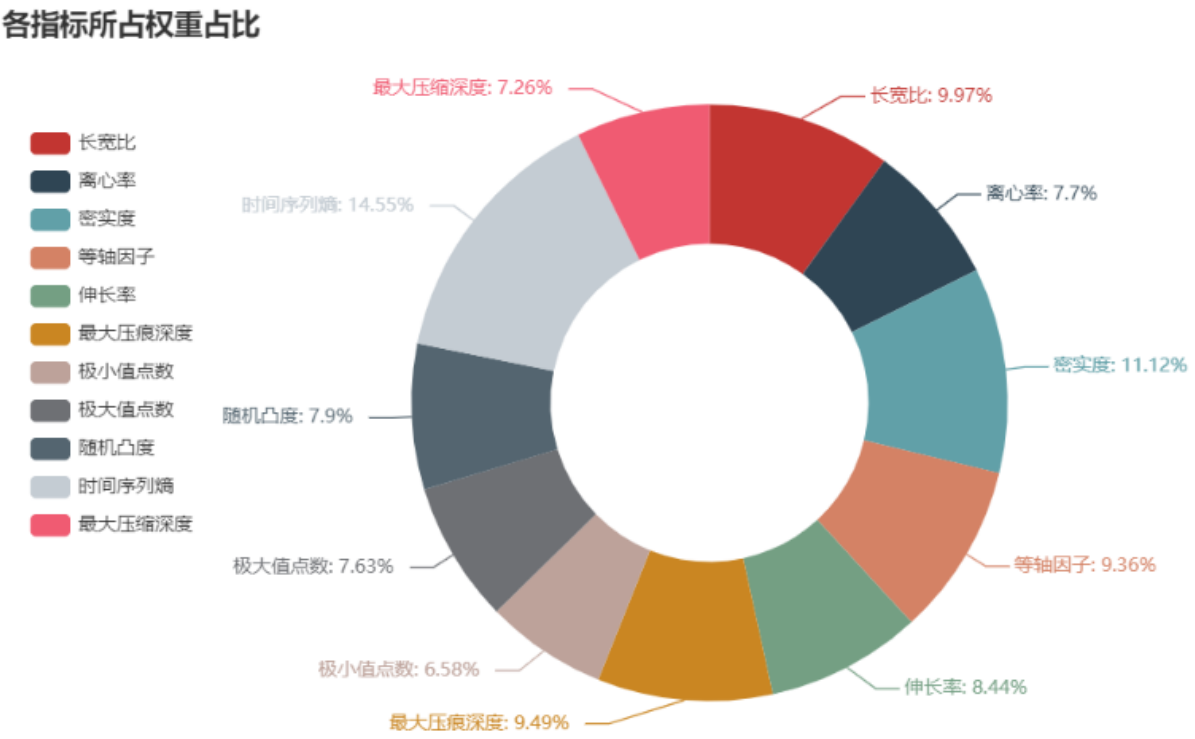


图 **8**　各维度权重占比图

5.3.1 模型性能评估

表 2 中，实验与三种神经网络进行纵向比较可知，四种神经网络训练后准确率都在 90% 左右，证实了神经网络非线性映射能力较强。Elman 网络和 DNN 网络的测试准确率都大于 BP 网络，说明反馈动态网络的逼近能力要强于前反馈静态网络。使用 PSO 优化 DNN 网络后，模型的跟踪性能有所改善，测试集中损失 (Validation Loss) 仅为 0.458，说明 PSO-DNN 网络的泛化能力和稳定性明显提高，且全局收敛性增强。

表 **2**　常见神经网络的比较

网络类型	网络名称	测试集中准确率
静态网络	BP	87.424%
静态网络	Elman	90.541%
动态网络	DNN	89.912%
动态网络	PSO-DNN	91.037%

六、 问题三模型的建立与求解

6.1 问题描述与分析

针对问题三，本题要求结合给出的叶子纹理的数据信息，对问题二原有模型进行改进并进行分析比较。本组基于问题二模型中原有的 11 维数据，加入叶子纹理的数据信息后增加网络输入层个数，修改 PSO-DNN 网络的参数，从而提高了叶片识别与分类的精度。

6.2 模型的建立与求解

加入叶片纹理数据后，根据第二问中模型原理进行参数调整，进行树叶分类预测的仿真实验：随机选取 80% 的数据作为训练集进行数据**标准化处理**，再对树叶种类进行**One-Hot 编码**作为分类目标。训练输入样本为 $1280*75$，测试输入样本为 $320*75$，DNN 网络输入层神经元 75 个，输出层神经元 100 个，包含 2 个隐含层，适应度函数选择均方误差。具体参数设置间表 3 。具体参数：

表 **3**　参数设置表

参数名称	参数符号	参数值
选择粒子个数为	T	30
计算精度	ε	0.00001
学习率	η	0.01
最大迭代次数	M	500
维度	S	75
加速因子 1	c_1	1.49
加速因子 2	c_2	1.49
惯性权重	β	0.9

通过上述参数设置，训练后求得最小均方误差为 0.253，**预测准确率为** 96.634%，其迭代过程中损失与准确率如下图所示：

图 **9**　**PSO-DNN** 迭代过程中损失与准确率

6.3 结果分析

分别与问题二中的 PSO 优化前后 DNN 模型进行比较，其结果如下所示：

表 **4**　新旧 **PSO-DNN** 网络的比较

网络模型	Validation Loss	Validation Accuracy
新 PSO-DNN 网络	0.253	96.634%
旧 PSO-DNN 网络	0.458	91.037%
新 DNN 网络	0.291	95.142%
旧 DNN 网络	0.518	89.912%

结果表明，问题三的 PSO-DNN 网络在测试集中准确率高于问题二中网络，说明在加入叶子纹理数据后，其分类模型预测能力升高，符合实际情况。与未经过 PSO 优化权值的网络比较均提升 1% 左右，证实了 PSO-DNN 网络非线性映射能力较强。

七、 模型的评价

7.1 模型的优点

7.2 模型的缺点

参考文献

[1] Tan Jing Wei,Chang Siow-Wee,Binti Abdul Kareem Sameem,Yap Hwa Jen,Yong Kien-Thai. Deep Learning for Plant Species Classification using Leaf Vein Morphometric.[J]. IEEE/ACM transactions on computational biology and bioinformatics,2018.

[2] Liu Jing,Sun Wanning,Su Yuting,Jing Peiguang,Yang Xiaokang. BE-CALF: Bit-Depth Enhancement by Concatenating All Level Features of DNN.[J]. IEEE transactions on image processing : a publication of the IEEE Signal Processing Society,2019,28(10).

[3] Matheus B. Vicari,Mathias Disney,Phil Wilkes,Andrew Burt,Kim Calders,William Woodgate. Leaf and wood classification framework for terrestrial LiDAR point clouds[J]. Methods in Ecology and Evolution,2019,10(5).

[4] 田德红, 何建敏. 基于变异粒子群优化与深度神经网络的航空弹药消耗预测模型 [J]. 南京理工大学学报,2018,42(06).

[5] 原继东, 王志海, 韩萌, 游洋. 基于逻辑 shapelets 转换的时间序列分类算法 [J]. 计算机学报,2015,38(07):1448-1459.

[6] 杨志辉, 胡红萍, 白艳萍. 基于主成分分析和 PSO-SVM 的树叶分类方法研究 [J]. 数学的实践与认识,2016,46(18):170-175.

[7] 侯铜, 姚立红, 阚江明. 基于叶片外形特征的植物识别研究 [J]. 湖南农业科学,2009(04):123-125+129.

[8] Febri Liantoni,Rifki Indra Perwira,Syahri Muharom,Riza Agung Firmansyah,Akhmad Fahruzi. Leaf classification with improved image feature based on the seven moment invariant[J]. Journal of Physics: Conference Series,2019,1175(1).

附录 A 代码

A.1 图片名称读取–matlab 源代码

```
p = genpath('.\data');% 获得文件夹data下所有子文件的路径，这些路径存在字符串p中，以';'分割
length_p = size(p,2);%字符串p的长度
path = {};%建立一个单元数组，数组的每个单元中包含一个目录
temp = [];
for i = 1:length_p %寻找分割符';'，一旦找到，则将路径temp写入path数组中
if p(i) ~= ';'
temp = [temp p(i)];
else
temp = [temp '\']; %在路径的最后加入 '\'
path = [path ; temp];
temp = [];
end
end
clear p length_p temp;
%至此获得data文件夹及其所有子文件夹（及子文件夹的子文件夹）的路径，存于数组path中。
%下面是逐一文件夹中读取图像
file_num = size(path,1);% 子文件夹的个数
for i = 1:file_num
file_path = path{i}; % 图像文件夹路径
img_path_list = dir(strcat(file_path,'*.jpg'));
img_num = length(img_path_list); %该文件夹中图像数量
if img_num > 0
for j = 1:img_num
image_name = img_path_list(j).name;% 图像名
%image = imread(strcat(file_path,image_name));
fprintf('%d %d %s\n',i-1,j,strcat(file_path,image_name));% 显示正在处理的路径和图像名
%图像处理过程 省略
%Untitled2(strcat(file_path,image_name));
end
end
end
```

A.2 树叶形状 id 计算–matlab 源代码

```
%UNTITLED2 Summary of this function goes here
%  Detailed explanation goes here
I=imread(str_path);
thresh = graythresh(I); %自动确定二值化阈值;
I1=im2bw(I);%二向化
stats = regionprops(I1,'Area','Eccentricity','Solidity','Perimeter', 'ConvexImage');
B = bwboundaries(I1,'noholes');
```

```
B=B{1,1};%边界坐标
A=stats.Area;%面积
P=stats.Perimeter;%周长
id(1)=stats.Eccentricity;%离心率
id(2)=stats.Solidity;%实密度
id(3)=4*pi*A/P.^2;%等周因子
temp1=zeros(size(B,1),size(B,1));
for i=1:size(B,1) %计算外接圆直径
for j=i:size(B,1)
temp1(i,j)=((B(i,1)-B(j,1)).^2+(B(i,2)-B(j,2)).^2).^0.5;
end
end
D=max(max(temp1)');%外接圆直径
temp2=zeros(size(B,1),1);
dd=zeros(size(I1,1),size(I1,2));
for i=1:size(I1,1)%计算内切圆半径
for j=1:size(I1,2)
if I1(i,j)==1
for k=1:size(B,1)
temp2(k,1)=((B(k,1)-i).^2+(B(k,2)-j).^2).^0.5;
end
dd(i,j)=min(temp2);
end
end
end
d=max(max(dd)');%内切圆半径
id(4)=1-2.*d./D;%伸长率
I3=stats.ConvexImage;
stats3 = regionprops(I3,'Centroid','Perimeter');
B3=bwboundaries(I3,'noholes');%凸型区域边界坐标
B3=B3{1,1};
C=stats3.Centroid;%凸型区域中心坐标
temp2=zeros(size(B3,1),1);
for k=1:size(B3,1)%计算最大距离
temp2(k,1)=((B3(k,1)-C(1,1)).^2+((B3(k,2)-C(1,2)).^2)).^0.5;
end
L1=max(temp2);%最大距离
temp2=zeros(size(B,1),1);
for k=1:size(B,1)%计算最大距离
temp2(k,1)=((B(k,1)-C(1,1)).^2+((B(k,2)-C(1,2)).^2)).^0.5;
end
L2=min(temp2);%最小距离
L=stats3.Perimeter;%凸型周长
id(5)=(L1-L2)./L;%最大压痕深度
for i=1:size(I1,1)
for j=1:size(I1,2)
if I1(i,j)==1
```

```
y1=i;
break;
end
end
end
for i=size(I1,1):-1:1
for j=size(I1,2):-1:1
if I1(i,j)==1
y2=i;
break;
end
end
end
for i=1:size(I1,2)
for j=1:size(I1,1)
if I1(j,i)==1
x1=i;
break;
end
end
end
for i=size(I1,2):-1:1
for j=size(I1,1):-1:1
if I1(j,i)==1
x2=i;
break;
end
end
end
id(6)=(x1-x2)./(y1-y2);%长宽比
fid=fopen('aaa.txt','a');%写入文件路径
fprintf(fid,'%f,%f,%f,%f,%f,%f\r\n',id);
fclose(fid);
end
```

A.3 时间序列展开–python 源代码

```
import pandas as pd

import scipy as sp
import scipy.ndimage as ndi
from scipy.signal import argrelextrema

import pandas as pd
```

```
from skimage import measure
from sklearn import metrics

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from pylab import rcParams
rcParams['figure.figsize'] = (6, 6)


# ------------------------------------------------------ I/O ---

def read_img(img_no,str):
"""reads image from disk"""
return mpimg.imread(str)


def get_imgs(num, str):
"""convenience function, yields random sample from leaves"""
if type(num) == int:
imgs = range(1, 1600)
num = np.random.choice(imgs, size=num, replace=False)

for img_no in num:
yield img_no, preprocess(read_img(img_no,str))


# ------------------------------------------------ preprocessing ---

def threshold(img, threshold=250):
"""splits img to 0 and 255 values at threshold"""
return ((img > threshold) * 255).astype(img.dtype)


def portrait(img):
"""makes all leaves stand straight"""
y, x = np.shape(img)
return img.transpose() if x > y else img


def resample(img, size):
"""resamples img to size without distorsion"""
ratio = size / max(np.shape(img))
return sp.misc.imresize(img, ratio, mode='L', interp='nearest')
```

```
def fill(img, size=500, tolerance=0.95):
"""extends the image if it is signifficantly smaller than size"""
y, x = np.shape(img)

if x <= size * tolerance:
pad = np.zeros((y, int((size - x) / 2)), dtype=int)
img = np.concatenate((pad, img, pad), axis=1)

if y <= size * tolerance:
pad = np.zeros((int((size - y) / 2), x), dtype=int)
img = np.concatenate((pad, img, pad), axis=0)

return img

# ------------------------------------------------- postprocessing ---

def standardize(arr1d):
"""move mean to zero, 1st SD to -1/+1"""
return (arr1d - arr1d.mean()) / arr1d.std()


def coords_to_cols(coords):
"""from x,y pairs to feature columns"""
return coords[::,1], coords[::,0]


def get_contour(img):
"""returns the coords of the longest contour"""
return max(measure.find_contours(img, .8), key=len)


def downsample_contour(coords, bins=512):
"""splits the array to ~equal bins, and returns one point per bin"""
edges = np.linspace(0, coords.shape[0],
num=bins).astype(int)
for b in range(bins-1):
yield [coords[edges[b]:edges[b+1],0].mean(),
coords[edges[b]:edges[b+1],1].mean()]


def get_center(img):
"""so that I do not have to remember the function ;)"""
return ndi.measurements.center_of_mass(img)

#------------------------------------------------- feature engineering ---

def extract_shape(img):
```

```
"""
Expects prepared image, returns leaf shape in img format.
The strength of smoothing had to be dynamically set
in order to get consistent results for different sizes.
"""
size = int(np.count_nonzero(img)/1000)
brush = int(5 * size/size**.75)
return ndi.gaussian_filter(img, sigma=brush, mode='nearest') > 200


def near0_ix(timeseries_1d, radius=5):
"""finds near-zero values in time-series"""
return np.where(timeseries_1d < radius)[0]


def dist_line_line(src_arr, tgt_arr):
"""
returns 2 tgt_arr length arrays,
1st is distances, 2nd is src_arr indices
"""
return np.array(sp.spatial.cKDTree(src_arr).query(tgt_arr))


def dist_line_point(src_arr, point):
"""returns 1d array with distances from point"""
point1d = [[point[0], point[1]]] * len(src_arr)
return metrics.pairwise.paired_distances(src_arr, point1d)


def index_diff(kdt_output_1):
"""
Shows pairwise distance between all n and n+1 elements.
Useful to see, how the dist_line_line maps the two lines.
"""
return np.diff(kdt_output_1)


# ------------------------------------------------------ wrapping functions ---

# wrapper function for all preprocessing tasks
def preprocess(img, do_portrait=True, do_resample=500,
do_fill=True, do_threshold=250):
""" prepares image for processing"""
if do_portrait:
img = portrait(img)
if do_resample:
img = resample(img, size=do_resample)
```

```
if do_fill:
img = fill(img, size=do_resample)
if do_threshold:
img = threshold(img, threshold=do_threshold)

return img


# wrapper function for feature extraction tasks
def get_std_contours(img):
"""from image to standard-length countour pairs"""

# shape in boolean n:m format
blur = extract_shape(img)

# contours in [[x,y], ...] format
blade = np.array(list(downsample_contour(get_contour(img))))
shape = np.array(list(downsample_contour(get_contour(blur))))

# flagging blade points that fall inside the shape contour
# notice that we are loosing subpixel information here
blade_y, blade_x = coords_to_cols(blade)
blade_inv_ix = blur[blade_x.astype(int), blade_y.astype(int)]

# img and shape centers
shape_cy, shape_cx = get_center(blur)
blade_cy, blade_cx = get_center(img)

# img distance, shape distance (for time series plotting)
blade_dist = dist_line_line(shape, blade)
shape_dist = dist_line_point(shape, [shape_cx, shape_cy])

# fixing false + signs in the blade time series
blade_dist[0, blade_inv_ix] = blade_dist[0, blade_inv_ix] * -1

return {'shape_img': blur,
'shape_contour': shape,
'shape_center': (shape_cx, shape_cy),
'shape_series': [shape_dist, range(len(shape_dist))],
'blade_img': img,
'blade_contour': blade,
'blade_center': (blade_cx, blade_cy),
'blade_series': blade_dist,
'inversion_ix': blade_inv_ix}


# !/usr/bin/python
```

```
# -*- coding:utf-8 -*-


import os

outer_path = r'C:\Users\77526\PycharmProjects\untitled\Demo5\data'
folderlist = os.listdir(outer_path) # 列举文件夹


for folder in folderlist:
inner_path = os.path.join(outer_path, folder)
total_num_folder = len(folderlist) # 文件夹的总数
filelist = os.listdir(inner_path) # 列举图片
i = 0
for item in filelist:
total_num_file = len(filelist) # 单个文件夹内图片的总数
if item.endswith('.jpg'):
src = os.path.join(os.path.abspath(inner_path), item) # 原图的地址
# dst = os.path.join(os.path.abspath(inner_path), str(folder) + '_' + str(
#     i) + '.jpg') # 新图的地址（这里可以把str(folder) + '_' + str(i) + '.jpg'改成你想改的名称）
try:
# os.rename(src, dst)
print(src)
i += 1


title, img = list(get_imgs([968], src))[0]
features = get_std_contours(img)
print(features['blade_series'])
df = pd.DataFrame(features['blade_series'][0]).T
# df.to_csv('blade_series.csv', mode='a', header=False, index=False)
plt.plot(*coords_to_cols(features['shape_contour']))
plt.plot(*coords_to_cols(features['blade_contour']))
#plt.axis('equal')

plt.subplot(122)
plt.plot(*features['shape_series'])
plt.plot(*features['blade_series'])
plt.show()
except:
continue



# plt.plot(*coords_to_cols(features['shape_contour']))
# plt.plot(*coords_to_cols(features['blade_contour']))
# #plt.axis('equal')
```

```
#
# plt.subplot(122)
# plt.plot(*features['shape_series'])
# plt.plot(*features['blade_series'])
# plt.show()
```

A.4 边缘测试时间序列–C++ 源代码

```
#include "stdafx.h"
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <string>
#include <io.h>
#include <vector>
#include <iostream>
#include <math.h>

using namespace cv;
using namespace std;
char * filePath = "C:\\Users\\77526\\PycharmProjects\\untitled\\Demo5\\data";
//此处用的是斜杠，也可以用反斜
//但需注意的是由于C语言的特点，要用双反斜杠，即"E:\\MATLAB\\LBP\\scene_categories"
//cin >> folder; //也可以用此段代码直接在DOS窗口输入地址，此时只需正常的单反斜杠即可

using namespace std;

void getFiles(string foler, vector<string>& files);

int main() {
string folder = filePath;

vector<string> files;
getFiles(folder, files ); //files为返回的文件名构成的字符串向量组

for( int i = 0; i < files.size(); i++ ) { //files.size()返回文件数量
//To do here
get_arr(files[i]);
cout << files[i] << endl;
}
system("pause");
return 0;
}
```

```
void getFiles( string path, vector<string>& files ) {
//文件句柄
long hFile =  0;
//文件信息
struct _finddata_t fileinfo; //大家可以去查看一下_finddata结构组成
//以及_findfirst和_findnext的用法，了解后妈妈就再也不用担心我以后不会编了
string p;
if((hFile = _findfirst(p.assign(path).append("\\*").c_str(),&fileinfo)) != -1) {
do {
//如果是目录,迭代之
//如果不是,加入列表
if((fileinfo.attrib & _A_SUBDIR)) {
if(strcmp(fileinfo.name,".") != 0 && strcmp(fileinfo.name,"..") != 0)
getFiles( p.assign(path).append("\\").append(fileinfo.name), files );
}
else {
files.push_back(p.assign(path).append("\\").append(fileinfo.name) );
}
}
while(_findnext(hFile, &fileinfo) == 0);_findclose(hFile);
}
}



void get_arr(string str){
//读取图像
Mat src_image = imread(str);
//图像读取出错处理
if (!src_image.data)
{
cout << "src image load failed!" << endl;
return -1;
}
//显示源图像
namedWindow("原图", WINDOW_NORMAL);
imshow("原图", src_image);

//此处高斯去燥有助于后面二值化处理的效果
//Mat blur_image;
//GaussianBlur(src_image, blur_image, Size(15, 15), 0, 0);
//imshow("GaussianBlur", blur_image);

/*灰度变换与二值化*/
Mat gray_image, binary_image;
cvtColor(src_image, gray_image, COLOR_BGR2GRAY);
threshold(gray_image, binary_image, 30, 255, THRESH_BINARY | THRESH_TRIANGLE);
imshow("binary", binary_image);
```

```
/*形态学闭操作*/
Mat morph_image;
Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3), Point(-1, -1));
morphologyEx(binary_image, morph_image, MORPH_CLOSE, kernel, Point(-1, -1), 2);
imshow("morphology", morph_image);

/*查找外轮廓*/
vector< vector<Point> > contours;
vector<Vec4i> hireachy;
findContours(binary_image, contours, hireachy, CV_RETR_EXTERNAL, CHAIN_APPROX_NONE, Point());

int l;//目标轮廓索引
//寻找最大轮廓，即目标轮廓
for (size_t t = 0; t < contours.size(); t++)
{
/*过滤掉小的干扰轮廓*/
Rect rect = boundingRect(contours[t]);
if (rect.width < src_image.cols / 2)
continue;
//if (rect.width >(src_image.cols - 20))
l = t;//找到了目标轮廓，获取轮廓的索引
}
//画出目标轮廓
Mat result_image = Mat::zeros(src_image.size(), CV_8UC3);
vector< vector<Point> > draw_contours;
draw_contours.push_back(contours[l]);
drawContours(result_image, draw_contours, -1, Scalar(255,255,255), 1, 8, hireachy);
namedWindow("lunkuo", WINDOW_NORMAL);
imshow("lunkuo", result_image);

//计算轮廓的傅里叶描述子
Point p;
int x, y, s;
int i = 0,j = 0,u=0;
s = (int)contours[l].size();
Mat src1(Size(s,1),CV_8SC2);
float f[9000];//轮廓的实际描述子
float fd[16];//归一化后的描述子，并取前15个
for (u = 0; u < s; u++)
{
float sumx=0, sumy=0;
for (j = 0; j < s; j++)
{
p = contours[l].at(j);
x = p.x;
y = p.y;
```

```
sumx += (float)(x*cos(2*CV_PI*u*j/s) + y*sin(2 * CV_PI*u*j / s));
sumy+= (float)(y*cos(2 * CV_PI*u*j / s) - x*sin(2 * CV_PI*u*j / s));
}
src1.at<Vec2b>(0, u)[0] = sumx;
src1.at<Vec2b>(0, u)[1] = sumy;
f[u] = sqrt((sumx*sumx)+(sumy*sumy));
}
//傅立叶描述字的归一化
f[0] = 0;
fd[0] = 0;
for (int k = 2; k < 17; k++)
{
f[k] = f[k] / f[1];
fd[k - 1] = f[k];
cout << fd[k-1] << endl;
}
//保存数据
for (int k = 0; k < 16; k++)
{
FILE *fp = fopen("1.txt", "a");
fprintf(fp, "%8f,", fd[k]);
fclose(fp);
}
FILE *fp = fopen("1.txt", "a");
fprintf(fp, "\n");
fclose(fp);
waitKey();
}
```

A.5 数据可视化–python 源代码

```
from pyecharts import options as opts
from pyecharts.charts import Page, Pie


def pie_base(list) -> Pie:
c = (
Pie()
.add("", list, radius=["40%", "75%"],)
.set_global_opts(title_opts=opts.TitleOpts(title="各指标所占权重占比"),
legend_opts=opts.LegendOpts(
orient="vertical", pos_top="15%", pos_left="2%"

),          toolbox_opts=opts.ToolboxOpts(),
```

```
)
.set_series_opts(label_opts=opts.LabelOpts(formatter="{b}: {d}% "))

)
return c

def pie_rich_label(list) -> Pie:
c = (
Pie()
.add(
"",list,
radius=["40%", "75%"],
label_opts=opts.LabelOpts(
position="outside",
formatter=" {b|{b}: }{per|{d}%} ",
background_color="#eee",
border_color="#aaa",
border_width=1,
border_radius=4,
rich={
"a": {"color": "#999", "lineHeight": 22, "align": "center"},
"abg": {
"backgroundColor": "#e3e3e3",
"width": "100%",
"align": "right",
"height": 22,
"borderRadius": [4, 4, 0, 0],
},
"hr": {
"borderColor": "#aaa",
"width": "100%",
"borderWidth": 0.5,
"height": 0,
},
"b": {"fontSize": 16, "lineHeight": 33},
"per": {
"color": "#eee",
"backgroundColor": "#334455",
"padding": [2, 4],
"borderRadius": 2,
},
},
),
)
.set_global_opts(title_opts=opts.TitleOpts(title="各指标所占权重占比"),legend_opts=opts.LegendOpts(
orient="vertical", pos_top="15%", pos_left="2%"
```

```
),          toolbox_opts=opts.ToolboxOpts(),
)
)
return c

list = [['长宽比',0.294],
['离心率',0.227],
['密实度',0.328],
['等轴因子',0.276],
['伸长率',0.249],
['最大压痕深度',0.280],
['极小值点数',0.194],
['极大值点数',0.225],
['随机凸度',0.233],
['时间序列熵',0.429],
['最大压缩深度',0.214]]

c = pie_base(list)
c.render("xs.html")
d = pie_rich_label(list)
d.render("xss.html")
```

A.6 PSO-DNN 网络搭建–python 源代码

```
import pandas as pd
import numpy as np
import os

trainData = pd.read_csv(r'C:\Users\77526\Downloads\all_data.csv')
trainData = trainData.iloc[:,1:]
print(trainData.head())

from sklearn.utils import shuffle
trainData = shuffle(trainData)
trainData = trainData.values
y = trainData[:,0:1]
X= trainData[:,1:].astype(float)

y=pd.DataFrame(y, columns=['species'])
df = pd.get_dummies(y,columns=['species'])
species = [s.replace('species_', '') for s in df.columns.tolist()]
df.columns = species

y = df.values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

from keras.models import Sequential #to initialize the neural network
from keras.layers import Dense # to build the layers of ANN
from keras.layers import Dropout


# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 100, kernel_initializer ='uniform', activation = 'relu',
    input_dim = 192))
classifier.add(Dropout(0.2))

# Adding the second hidden layer
classifier.add(Dense(units = 100, kernel_initializer ='uniform', activation = 'relu'))
classifier.add(Dropout(0.2))

# Adding the third hidden layer
classifier.add(Dense(units = 100, kernel_initializer ='uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units=99, kernel_initializer = 'uniform', activation = 'softmax'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
    ['accuracy'])

# Fitting the ANN to the Training set
model = classifier.fit(X_train, y_train, batch_size = 5, nb_epoch = 500)
print(model)


## Plotting the error with the number of iterations
## With each iteration the error reduces smoothly
import matplotlib.pyplot as plt
# define the function
def training_vis(hist):
loss = hist.history['loss']
val_loss = hist.history['val_loss']
```

```
acc = hist.history['acc']
val_acc = hist.history['val_acc']

# make a figure
fig = plt.figure(figsize=(8,4))
# subplot loss
ax1 = fig.add_subplot(121)
ax1.plot(loss,label='train_loss')
ax1.plot(val_loss,label='val_loss')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Loss')
ax1.set_title('Loss on Training and Validation Data')
ax1.legend()
# subplot acc
ax2 = fig.add_subplot(122)
ax2.plot(acc,label='train_acc')
ax2.plot(val_acc,label='val_acc')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Accuracy')
ax2.set_title('Accuracy on Training and Validation Data')
ax2.legend()
plt.tight_layout()
plt.show()



# call the function
training_vis(hist)
loss1 = hist.history['loss']
val_loss1 = hist.history['val_loss']
acc1 = hist.history['acc']
val_acc1 = hist.history['val_acc']




#计算权重
# loss,accuracy=classifier.evaluate(X_test,y_test)

# print('loss:',loss)
# print('accuracy:',accuracy)
weight_Dense_2,bias_Dense_3 = classifier.get_layer('input').get_weights()
weight_Dense_1,bias_Dense_3 = classifier.get_layer('output').get_weights()
df1 = pd.DataFrame(weight_Dense_1)
df2 = pd.DataFrame(weight_Dense_2)
df = df2.mul(df1,axis='columns',level=None, fill_value=None)
# df2.shape
x= abs(df2[1])
su =0
```

```
for i in range(0,15):
x= abs(df2.T[i])
su =su+sum(x)
print(sum(x)/100)
print(su)
```

A.7 各类分类器比较–python 源代码

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def warn(*args, **kwargs): pass
import warnings
warnings.warn = warn

from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import StratifiedShuffleSplit

train = pd.read_csv(r'C:\Users\77526\Downloads\all_data.csv')
# test = pd.read_csv('./test.csv')

def encode(train):
le = LabelEncoder().fit(train.species) #对数据进行标签编码
labels = le.transform(train.species)     # encode species strings
classes = list(le.classes_)              # save column names for submission
#     test_ids = test.id                       # save test ids for submission

train = train.drop(['species', 'id'], axis=1)
#     test = test.drop(['id'], axis=1)

return train, labels, classes

train, labels, classes = encode(train)
sss = StratifiedShuffleSplit(labels, 10, test_size=0.2, random_state=23)

for train_index, test_index in sss:
X_train, X_test = train.values[train_index], train.values[test_index]
y_train, y_test = labels[train_index], labels[test_index]

from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
```

```
    GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

classifiers = [
KNeighborsClassifier(3),
SVC(kernel="rbf", C=0.025, probability=True),
NuSVC(probability=True),
DecisionTreeClassifier(),
RandomForestClassifier(),
GradientBoostingClassifier(),
GaussianNB(),
LinearDiscriminantAnalysis(),]

<<<<<<< HEAD




=======
# Logging for Visual Comparison
log_cols=["Classifier", "Accuracy", "Log Loss"]
log = pd.DataFrame(columns=log_cols)

for clf in classifiers:
clf.fit(X_train, y_train)
name = clf.__class__.__name__

print("="*30)
print(name)

print('****Results****')
train_predictions = clf.predict(X_test)
acc = accuracy_score(y_test, train_predictions)
print("Accuracy: {:.4%}".format(acc))

train_predictions = clf.predict_proba(X_test)
ll = log_loss(y_test, train_predictions)
print("Log Loss: {}".format(ll))

log_entry = pd.DataFrame([[name, acc*100, ll]], columns=log_cols)
log = log.append(log_entry)

print("="*30)
sns.set_color_codes("muted")
sns.barplot(x='Accuracy', y='Classifier', data=log, color="b")
```

```
plt.xlabel('Accuracy %')
plt.title('Classifier Accuracy')
plt.show()

sns.set_color_codes("muted")
sns.barplot(x='Log Loss', y='Classifier', data=log, color="g")

plt.xlabel('Log Loss')
plt.title('Classifier Log Loss')
plt.show()
```