数据分析

# 0 准备

## 0.1 数据来源

使用UCI Machine Learning Repository提供的Dry Bean Dataset数据集，链接为 : <https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset>

获取方式为直接下载，数据集下载连接为: <https://archive.ics.uci.edu/ml/machine-learning-databases/00602/DryBeanDataset.zip>

网站提供了关于数据的简要叙述:

本研究使用了七种不同类型的干豆，根据市场情况考虑了形状、形状、类型和结构等特征。为了获得统一的种子分类，开发了一种计算机视觉系统来区分具有相似特征的七种不同注册的干豆品种。对于分类模型，使用高分辨率相机拍摄了7种不同注册干豆的13611粒图像。通过计算机视觉系统获得的豆图像经过分割和特征提取阶段，共有16个特征；12个尺寸和4个形状。

这一开源数据的来源为Koklu及Ozkan等人在2020年发表的文章，该文章通过使用高分辨率相机拍摄豆子，并使用机器学习来计算豆子的各相形状参数。

KOKLU, M. and OZKAN, I.A., (2020), â€œMulticlass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques.â€? Computers and Electronics in Agriculture, 174, 105507.

数据文件包含了描述的文本文件Dry\_Bean\_Dataset.txt 及数据文件Dry\_Bean\_Dataset.xlsx 、Dry\_Bean\_Dataset.arff 。arff文件和exls文件都是数据文件，内容相同。

## 0.2 依赖

利用python 3.9.4，结合jupyter notebook实现数据读取，分析、建立模型并分析分类结果等任务，以下是依赖的第三方库:

import numpy, pandas, seaborn  
from matplotlib import pyplot as plt  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn import metrics  
复制

# 1.数据初步分析

## 1.1 数据总体情况

利用python 的第三方库pandas从Dry\_Bean\_Dataset.xlsx 中读入数据，使用函数pandas.read\_excel()

DATASET\_PATH = "Dry\_Bean\_Dataset.xlsx"  
data = pandas.read\_excel(DATASET\_PATH)  
复制

查看数据条目:

data.info()  
复制

Pasted image 20221128104011.png

* 这显示数据具有17项特征值，其中数值有16项(小数14项， 整数2项)，字符串型1项。
* 显示数据条目共有13611条

## 1.2 数据特征的分布

可以使用DataFrame.head(N)查看数据的前N条

data.head(10)  
复制

结合1.1中的数据特征可推知前16项特征是描述豆子的参数，最后一项 Class 是豆子的类别

### 1.2.1 分析豆子类别的分布

可以对Class特征使用value\_counts()方法来检查豆子类别的分布

data['Class'].value\_counts()  
复制

Pasted image 20221128104940.png

* 结果表明，DERMASON 类豆子最多，共3546条数据，BOMBAY 类豆子最少共522条数据。  
  使用直方图可以直观看到数据分布

data\_classes\_distribution = data['Class'].value\_counts()  
plt.figure(figsize=(10, 7))  
plt.title('数据集豆子种类分布')  
  
colors=['tomato','orange','gold','darkseagreen','turquoise','skyblue','plum']  
plt.bar(data\_classes\_distribution.index, data\_classes\_distribution, color=colors)  
for i,j in zip(data\_classes\_distribution.index, data\_classes\_distribution):  
  
 plt.text(i, j+5, '{:.0f}'.format(j), ha='center', va= 'bottom',fontsize=13)  
复制

Bean\_Class\_Distribution 2.jpg

### 1.2.2 描述特征的分布

使用DataFrame.describe()来查看数据均值、标准差、最值、不同的分位数

data.describe()  
复制

Pasted image 20221128112317.png

* 可以使用matploylib.pyplot.boxplot() 来观察箱型图。

fig = plt.figure(figsize=(16, 16))  
plt.suptitle('特征箱型图(未显示异常值)')  
for i in range(16):  
 ax = fig.add\_subplot(4, 4, i+1)  
 ax.boxplot(data.iloc[:, i], showfliers=False)   
 ax.set\_title(data.columns[i])  
   
复制

boxplot.jpg

* 使用DataFrame.hist()可以计算各特征的分布:

data.hist(bins = 50, figsize= (20,15))  
plt.show()  
复制

b.jpg

* 利用pandas.plotting.scatter\_matrix() 函数，可以对数据作散点矩阵，来检查数据相关性。

pandas.plotting.scatter\_matrix(data, figsize=(20, 20), marker='o', diagonal='kde', alpha=0.01)  
复制

c.jpg

* 可以使用DataFrame.corr()进行查看各特征相关矩阵，再结合matplotlib.pyplot.imshow()函数进行可视化。

corr\_matrix = data.corr()  
plt.figure(figsize=(16,16))  
plt.imshow(corr\_matrix.abs(), cmap= plt.cm.OrRd, vmax=1, vmin=0)  
thresh = 0.5  
for i, j in itertools.product(range(corr\_matrix.shape[0]), range(corr\_matrix.shape[1])):  
 plt.text(j, i, '{:.2f}'.format(corr\_matrix.iloc[i, j]), horizontalalignment="center",  
 color="white" if corr\_matrix.abs().iloc[i, j] > thresh else "black")  
  
  
tick\_marks = numpy.arange(len(corr\_matrix.index))  
plt.xticks(tick\_marks, corr\_matrix.index, rotation=45)  
plt.yticks(tick\_marks, corr\_matrix.index)  
plt.title('相关性矩阵')  
plt.colorbar(fraction=0.046)  
复制

Corr\_matrix 1.jpg

* 从以上图表上看， 特征Area 、Perimeter 、MajorAxisLength、 MinorAxisLenght 、ConvexArea、EquivDiameter 显示了较为相似的分布。这些特征分别描述了:
  + Area : 豆子的面积
  + Perimeter: 豆子的周长
  + Major axis length: 豆子的长轴长度
  + Minor axis length: 豆子的短轴长度
  + Convex area: 可以包含豆子种子的最小凸多边形面积
  + Equivalent diameter: 等同于豆子面积的圆的直径
  + 上述特征描述了豆子大小的度量，并显示出了较高的相关性。
* 从图表上看，特征Aspect ratio、Eccentricity、Compactness 、ShapeFactor3 显示了较为相似的分布。这些特征分别描述了:
  + Aspect ratio: 纵横比，定义为长短轴的比
  + Eccentricity: 具有与区域相同力矩的椭圆的偏心率
  + Compactness: 豆子的圆度， 为Equivalent diameter/Major axis length
  + ShapeFactor3: 数据未作说明
  + 上述特征描述豆子形状，显示了较高的相关性。其中shapefactor3与compactness近似成线性关系。
* 其他特征包括了 solidity 、extent等，较独立于其他特征
* 从ShapeFactor1~4 的4组散点图可以发现，数据显著聚为两个大类。

选择两个组中具有代表性的特征进行分析:

seaborn.scatterplot(data=data, x='ConvexArea', y='Compactness', alpha=0.2, hue='Class', hue\_order=class\_color.keys(), palette=class\_color.values())  
复制

d.jpg

* BOMBAY类豆子和其余六个种类区别较大， 对应了分布图中豆子面积较大的小峰。可以推测， BOMBAY是较大的豆子，根据大小可以容易地区分BOMBAY及其他豆子。

观察散点矩阵的Compactness-ShapeFactor1散点图，大的数据聚集内具有较为清晰的边界

seaborn.scatterplot(data=data, x='Compactness', y='ShapeFactor1', alpha=0.2, hue='Class', hue\_order=class\_color.keys(), palette=class\_color.values())  
复制

e.jpg

* 在Compactness及ShapeFactor维度，数据显示了更好的可分离性。

# 2. 决策树模型

* 由于需要从豆子的各相特征分析其类别，因此该问题属于分类问题。
* 由于豆子种类一共7种，因此该问题属于多分类问题。

可以使用决策树进行分类

## 2.1 预处理

由于豆子类别以文本方式给出，即Class=['BARBUNYA', 'BOMBAY', 'CALI', 'DERMASON', 'HOROZ', 'SEKER', 'SIRA'] ,属于计算机无法理解的标签。为了更好完成分类任务，需要将类别转换为独热向量:

# One-hot encoding  
labels = pandas.get\_dummies(data['Class'])  
复制

转换后的标签为:

labels  
复制

Pasted image 20221206103821.png

对于豆子特征，由于在数据初步分析时可以发现，描述豆子大小的特征均值和标准差较大，可以将其进行归一化，这里使用sklearn.preprocessing.StandardScaler() 进行归一化，使得数据均值为0， 标准差为1:

# Features  
scaler = StandardScaler()  
bean\_p = scaler.fit\_transform(data.iloc[:, 0:16])  
  
bean\_p  
复制

Pasted image 20221206110842.png

## 2.2 划分训练集和测试集

使用sklearn.model\_selection.train\_test\_split() 可以划分数据为训练集和测试集:

train\_X, test\_X, train\_Y, test\_Y = train\_test\_split(bean\_p, labels, test\_size = 0.2, random\_state = 1)  
复制

* 设置了测试集占整个数据集的0.2，而训练集为数据集的0.8
* 设置random\_state为1，在复现时可以再次设置为1得到相同的测试集/训练集划分

train\_X 训练集样本个数为10888，test\_X 测试集样本个数为2723：

>>> train\_X.shape  
(10888, 16)  
>>> train\_Y.shape  
(2723, 16)  
复制

训练集上的豆子类别分布:

plt.figure(figsize=(10, 7))  
colors=['tomato','orange','gold','darkseagreen','turquoise','skyblue','plum']  
train\_classes\_dis = pandas.Series(train\_Y.apply(sum, axis='index')).sort\_values(0,ascending=False)  
  
  
plt.title('训练集的类别分布')  
plt.bar(train\_classes\_dis.index, train\_classes\_dis, color=colors)  
for i,j in zip(train\_classes\_dis.index, train\_classes\_dis):  
 plt.text(i, j+5, '{:.0f}'.format(j), ha='center', va= 'bottom',fontsize=13)  
复制

train\_classes\_Distribution 1.jpg

## 2.3 利用决策树进行分类

可以使用适合多分类任务的决策树。sklearn库提供了建立决策树模型的函数:

sklearn.tree.DecisionTreeClassifier(\*, criterion='gini', splitter='best', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, class\_weight=None, ccp\_alpha=0.0)  
复制

可以通过控制上述函数形参的值来调整决策树参数

* criterion 可以设置损失函数的计算方式，可以选基尼系数或信息熵。由于两者算法对准确率影响不大，而信息熵由于需要进行对数运算，所以更慢，因此选择基尼系数。
* splitter 可以设置每个树节点的拆分策略。“best”选择最好的分割，“random”选择最好的随机分割。由于样本数量较大，因此选用"random"更合适。
* max\_depth 可以设置树深，设置太大会导致过拟合，设置太小会导致欠拟合。在调参过程中，从max\_depth=5 开始逐步检查模型效果，发现在max\_depth=10 时达到较好效果。
* class\_weight 可以设置样本给各类别的权重，主要是为了防止训练集某些类别的样本过多导致训练的决策树过于偏向这些类别。由于类别DERMASON 类豆子是BOMBAY 豆子数量的7倍之多，因此使用用“balanced”，令算法自己计算类别权重。

最终设定决策树为:

from sklearn import tree  
tree\_clf = tree.DecisionTreeClassifier(splitter='best',  
 class\_weight='balanced',  
 max\_depth=10)  
复制

可以使用.fit() 训练样本

tree\_clf.fit(train\_X, train\_Y)  
复制

## 2.4 训练结果

### 2.4.1 数值指标

sklearn.metrics.accuracy\_score() 函数计算给定真值与预测值的准确率。  
sklearn.metrics.precision\_score() 函数计算给定真值与预测值的精确率。  
sklearn.metrics.recall\_score() 函数计算给定真值与预测值的召回率。

训练集上的准确率、精确率、召回率:

>>> metrics.accuracy\_score(train\_Y, tree\_clf.predict(train\_X))  
0.949210139603233  
  
>>> metrics.precision\_score(train\_Y, tree\_clf.predict(train\_X), average='weighted')  
0.9501488500422913  
  
>>> metrics.recall\_score(train\_Y, tree\_clf.predict(train\_X), average='weighted')  
0.949210139603233  
复制

测试集上的准确率、精确率、召回率:

>>> metrics.accuracy\_score(test\_Y, tree\_clf.predict(test\_X))  
0.9162688211531399  
  
>>> metrics.precision\_score(test\_Y, tree\_clf.predict(test\_X), average='weighted')  
0.9185090650102509  
  
>>> metrics.recall\_score(test\_Y, tree\_clf.predict(test\_X), average='weighted')  
0.9162688211531399  
复制

### 2.4.2 混淆矩阵

编写混淆矩阵画图函数：

def plot\_confusion\_matrix(cm, classes,  
 title='Confusion matrix',  
 cmap=plt.cm.OrRd,  
 save\_path = None):  
 '''  
 cm: confusion matrix, classes: classes name  
 '''  
  
 cm = cm.astype('float') / cm.sum(axis=1)[:, numpy.newaxis] #计算百分比  
 plt.figure(figsize=(10,10))  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 tick\_marks = numpy.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
 thresh = cm.max() / 2.  
  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, '{:.2f}'.format(cm[i, j]), horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.colorbar(fraction=0.046)  
  
 if save\_path:  
 plt.savefig(save\_path, dpi = 300, bbox\_inches= 'tight', pad\_inches=0.2)  
  
 plt.show()  
  
def CM(model,test\_data,true\_labels):  
  
 '''  
 calculate the confusion matrix  
 '''  
 predictions = numpy.array(model.predict(test\_data))  
 predictions = predictions.argmax(axis=-1)  
 true\_labels = numpy.array(true\_labels).argmax(axis=-1)  
  
 return metrics.confusion\_matrix(y\_true=true\_labels, y\_pred=predictions)  
  
复制

* plot\_confusion\_matrix() 函数参与混淆矩阵作图及保存工作
* CM() 函数接受分类器、输入样本及目标标签，并根据预测结果使用sklearn.metrics.confusion\_matrix() 计算混淆矩阵(结果为正确样本数)

使用以下代码调用画出函数

cm\_train = CM(tree\_clf,train\_X,train\_Y)   
plot\_confusion\_matrix(cm\_train, classes = list(train\_Y.columns), title='训练集混淆矩阵', save\_path='confusion\_matrix\_training\_set.jpg')  
  
cm\_test = CM(tree\_clf,test\_X,test\_Y)   
plot\_confusion\_matrix(cm\_test, classes = list(test\_Y.columns), title='测试集混淆矩阵', save\_path='confusion\_matrix\_testing\_set.jpg')  
复制

得到训练集上的混淆矩阵:  
confusion\_matrix\_training\_set.jpg  
测试集上的混淆矩阵:  
confusion\_matrix\_testing\_set.jpg  
分析上图混淆矩阵可以发现，大部分类别都能较为准确地被分类， 但存在一些错分情况:

* DERMASON 和 SIRA 类容易错分
* CALI 和 BARBUNYA 类容易错分

plt.figure(figsize=(10, 8))  
seaborn.scatterplot(data=data, x='MajorAxisLength', y='ShapeFactor1', alpha=0.2, hue='Class', hue\_order=class\_color.keys(), palette=class\_color.values())  
复制

作图可以发现，容易错分的类别在较为重要的特征上可分性不佳。 DERMASON 和 SIRA 类(分别为红色和橙色) 存在较大混叠，CALI 和 BARBUNYA 类(分别为青和蓝) 也存在较大混叠。这些类别在特征上的混叠导致训练结果不及预期。  
MAL\_SH1.jpg

### 2.4.3 特征对分类结果的影响程度

sklearn 提供了返回特征对分类结果的方法 .feature\_importances\_():

df\_im = pandas.DataFrame([\*zip(test\_X.columns, tree\_clf.feature\_importances\_)])  
  
plt.figure(figsize=(12, 8))  
plt.title('特征对类别的影响')  
plt.bar(df\_im.iloc[:,0], df\_im.iloc[:,1])  
tick\_marks = numpy.arange(len(df\_im.iloc[:,0]))  
plt.xticks(tick\_marks, df\_im.iloc[:,0], rotation=45)  
for a,b in zip(tick\_marks, df\_im.iloc[:,1]):  
  
 plt.text(a, b+0.005, '{:.4f}'.format(b), ha='center', va= 'bottom',fontsize=11)  
  
plt.savefig('features\_importances.jpg', dpi = 300, bbox\_inches = 'tight', pad\_inches=0.2)  
复制

可以得到:  
features\_importances.jpg  
从图表中可看出，特征ShapeFactor1 对分类结果影响最高，最主要影响的五个特征为:

* ShapeFactor1
* ShapeFactor3
* MajorAxisLength
* Perimeter
* roundness

# 3 总结

1. 本次实验使用python作为编程语言，结合第三方库如pandas, numpy ,sklearn , matplotlib辅助实现数据读取、数据分析和可视化、分类模型的建立及分类评价。
2. Dry Bean Dataset数据集包含了共13611条数据，数据集有16个特征，豆子种类共7个。数据没有空值，特征的分布如第一部分所述，一些特征之间显示了较强的相关性。豆子数量最多的为DERMASON类豆子，数量最少的是BOMBAY类豆子，前者是后者的7倍左右。
3. 数据预处理过程中，对各特征进行了归一化和标准化，使其均值为0方差为1。将离散的类别特征转换为了独热编码向量。训练集和数据集随机划分，其数据量为4:1。
4. 针对此多分类任务，选择使用决策树对数据进行机器学习，并利用python第三方库sklearn中的决策树类sklearn.tree.DescisionTree做代码实现。根据数据集特征并经过调参，最终确定的决策树为使用基尼系数的、最大深度为10的、树节点分割方式为最好随机分割的、各类别权重自适应的决策树。最终得在测试集上的分类准确度为0.916，精确度为0.918，召回率0.916。
5. 根据混淆矩阵，可以发现类别 SIRA 和 DERMASON 类, CALI 和 BARBUNYA 容易出现分类错误。从特征分布散点图来看，以上两组类别可分性不佳，导致分类结果不及预期。分类结果较为准确的是 BOMBAY 类，其准确率为1， 从特征分布散点图上看，该类别与其他类别可分性较好。
6. 根据对决策树的分析，最能区分类别的前五个特征是 : ShapeFactor1， ShapeFactor3， MajorAxisLength， Perimeter， roundness
7. 形成了相关图表和数值作为实验成果。