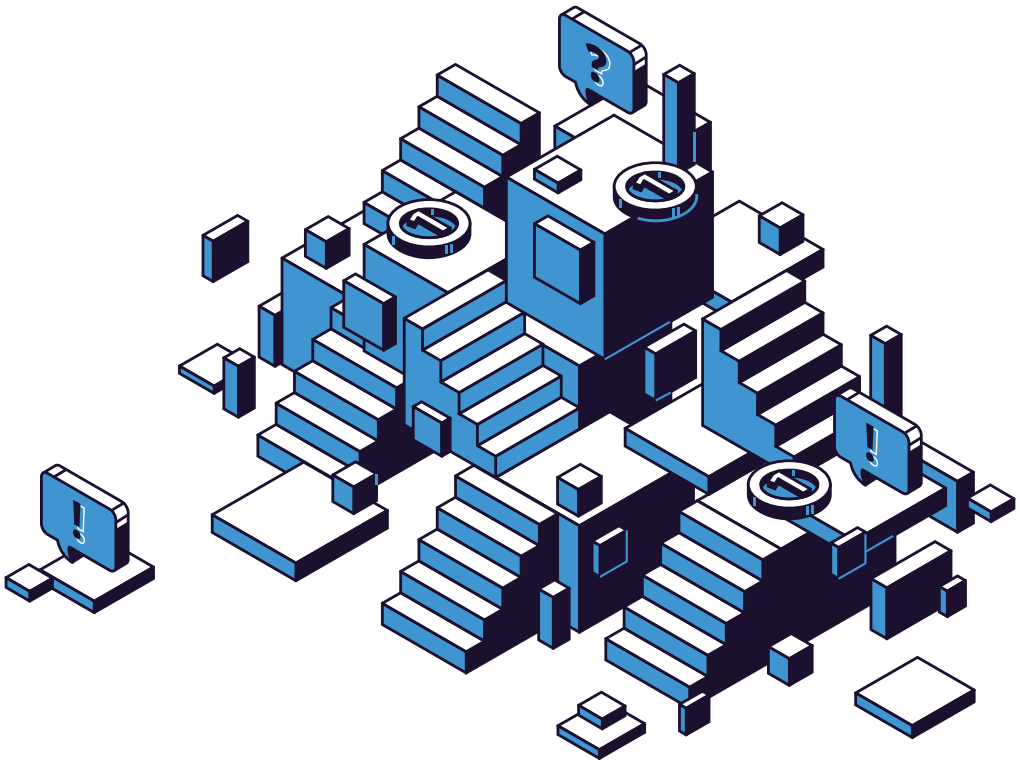
Datawhale

机器学习算法(六)

——基于决策树的分类预测



Datawhale 出品

前言

内容简介

由于决策树模型中自变量与因变量的非线性关系以及决策树简单的计算方法，使得它成为集成学习中最为广泛使用的基模型。梯度提升树(GBDT)，XGBoost以及LightGBM等集成模型都采用了决策树作为基模型，在广告计算、CTR预估、金融风控等领域大放异彩，成为当今与神经网络相提并论的复杂模型，更是数据挖掘比赛中的常客。在新的研究中，南京大学周志华老师提出一种多粒度级联森林模型，创造了一种全新的基于决策树的深度集成方法，为我们提供了决策树发展的另一种可能。

学习实践

本教程为大家详细讲解了基于逻辑回归的分类预测中涉及到的**知识细节**，并提供了**完整的代码流程**，甚至是可以完全**复现的代码内容**。

考虑到在Github的开源内容经常打不开，上述内容也开源在了国内阿里云社区。阿里云作为Datawhale开源支持方，将提供**免配置运行环境**和**免费算力**支持。

点击开始体验，即可学习实践。
https://developer.aliyun.com/ai/scenario/bb2fe211e5e94017840ce42cc31fe621?

Tips：平台内部的软硬件资源**完全免费**，大家也可以在上面运行自己的代码（薅羊毛）！

编著作者

小雨姑娘

开源支持方

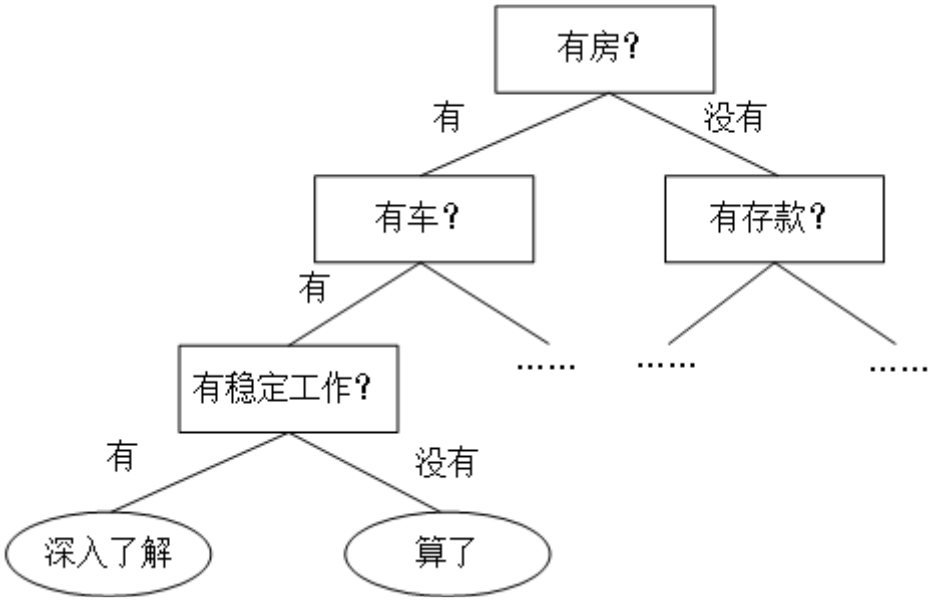
二维码

目录

1 决策树的介绍和应用

1.1 决策树的介绍

决策树是一种常见的分类模型，在金融分控、医疗辅助诊断等诸多行业具有较为广泛的应用。决策树的核心思想是基于树结构对数据进行划分，这种思想是人类处理问题时的本能方法。例如在婚恋市场中，女方通常会先看男方是否有房产，如果有房产再看是否有车产，如果有车产再看是否有稳定工作……最后得出是否要深入了解的判断。



决策树的主要优点：

1. 具有很好的解释性，模型可以生成可以理解的规则。
2. 可以发现特征的重要程度。
3. 模型的计算复杂度较低。

决策树的主要缺点：

1. 模型容易过拟合，需要采用减枝技术处理。
2. 不能很好利用连续型特征。
3. 预测能力有限，无法达到其他强监督模型效果。
4. 方差较高，数据分布的轻微改变很容易造成树结构完全不同。

1.2 决策树的应用

由于决策树模型中自变量与因变量的非线性关系以及决策树简单的计算方法，使得它成为集成学习中最为广泛使用的基模型。梯度提升树(GBDT)，XGBoost以及LightGBM等集成模型都采用了决策树作为基模型，在广告计算、CTR预估、金融风控等领域大放异彩，成为当今与神经网络相提并论的复杂模型，更是数据挖掘比赛中的常客。在新的研究中，南京大学周志华老师提出一种多粒度级联森林模型，创造了一种全新的基于决策树的深度集成方法，为我们提供了决策树发展的另一种可能。

同时决策树在一些需要明确可解释甚至提取分类规则的场景中被广泛应用，而其他机器学习模型在这一点很难做到。例如在医疗辅助系统中，为了方便专业人员发现错误，常常将决策树算法用于辅助病症检测。例如在一个预测哮喘患者的模型中，医生发现测试的许多高级模型的效果非常差。所以他们在数据上运行了一个决策树的模型，发现算法认为剧烈咳嗽的病人患哮喘的风险很小。但医生非常清楚剧烈咳嗽一般都会被立刻检查治疗，这意味着患有剧烈咳嗽的哮喘病人都会马上得到收治。用于建模的数据认为这类病人风险很小，是因为所有这类病人都得到了及时治疗，所以极少有人在此之后患病或死亡。

2 实验室手册

2.1 学习目标

1. 了解 决策树 的理论知识
2. 掌握 决策树 的 sklearn 函数调用使用并将其运用到企鹅数据集预测

2.2 代码流程

Part1 Demo实践

1. Step1:库函数导入
2. Step2:模型训练
3. Step3:数据和模型可视化
4. Step4:模型预测

Part2 基于企鹅（penguins）数据集的决策树分类实践

1. Step1:库函数导入
2. Step2:数据读取/载入
3. Step3:数据信息简单查看
4. Step4:可视化描述
5. Step5:利用 决策树模型 在二分类上 进行训练和预测
6. Step6:利用 决策树模型 在三分类(多分类)上 进行训练和预测

2.3 算法实战

2.3.1 Demo实践

2.3.1.1 Step1： 库函数导入

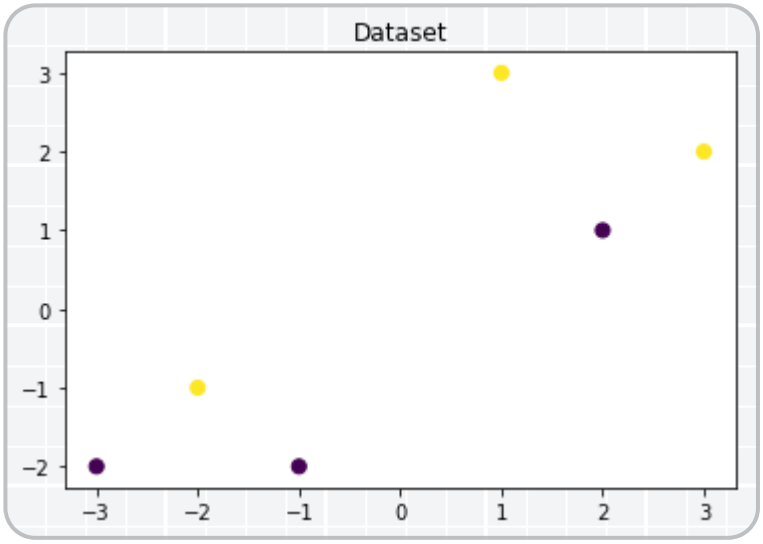
```
##  基础函数库
import numpy as np

## 导入画图库
import matplotlib.pyplot as plt
import seaborn as sns

## 导入决策树模型函数
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

2.3.1.2 Step2：训练模型

```
##Demo演示LogisticRegression分类

## 构造数据集
x_fearures = np.array([[-1, -2], [-2, -1], [-3, -2], [1, 3], [2, 1], [3, 2]])
y_label = np.array([0, 1, 0, 1, 0, 1])

## 调用逻辑回归模型
tree_clf = DecisionTreeClassifier()

## 用逻辑回归模型拟合构造的数据集
tree_clf = tree_clf.fit(x_fearures, y_label)
```

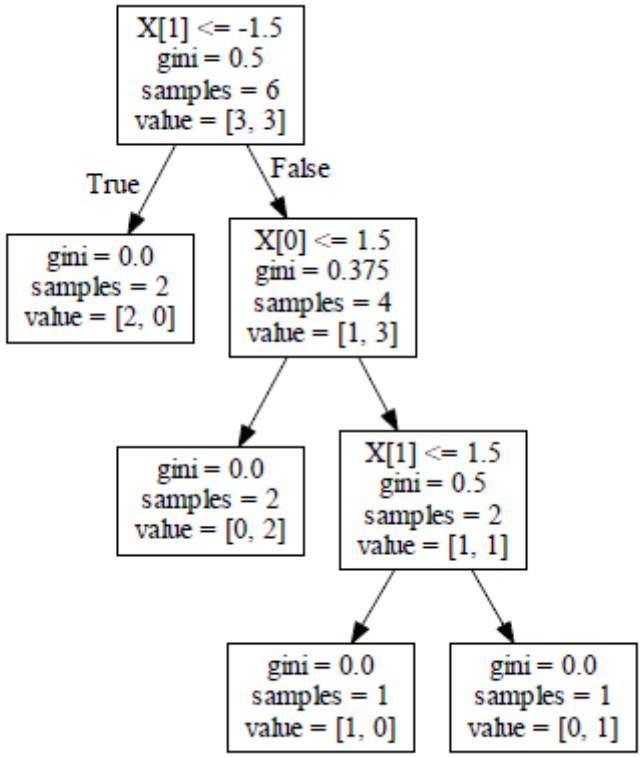
2.3.1.3 Step3：数据和模型可视化（需要用到graphviz可视化库）

```
## 可视化构造的数据样本点
plt.figure()
plt.scatter(x_fearures[:,0],x_fearures[:,1], c=y_label, s=50, cmap='viridis')
plt.title('Dataset')
plt.show()
```




```
## 可视化决策树
import graphviz
dot_data = tree.export_graphviz(tree_clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("pengunis")
```

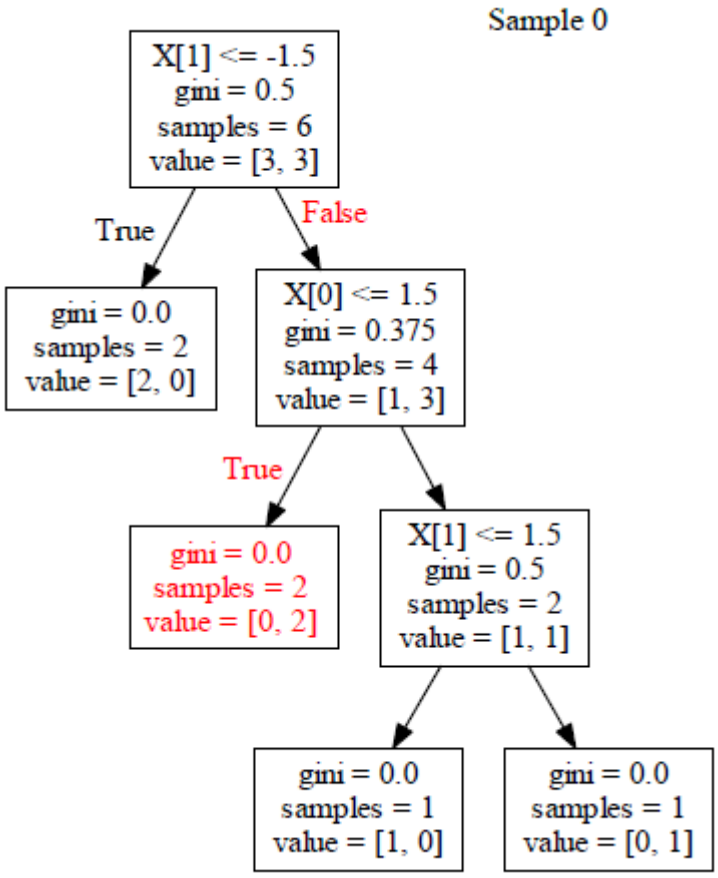
```
'pengunis.pdf'
```



2.3.1.4 Step4：模型预测

```
## 创建新样本
x_fearures_new1 = np.array([[0, -1]])
x_fearures_new2 = np.array([[2, 1]])

## 在训练集和测试集上分布利用训练好的模型进行预测
y_label_new1_predict = tree_clf.predict(x_fearures_new1)
y_label_new2_predict = tree_clf.predict(x_fearures_new2)

print('The New point 1 predict class:\n',y_label_new1_predict)
print('The New point 2 predict class:\n',y_label_new2_predict)
```

```
The New point 1 predict class:
 [1]
The New point 2 predict class:
 [0]
```



2.3.2 基于企鹅数据集的决策树实战

在实践的最开始，我们首先需要导入一些基础的函数库包括：numpy （Python进行科学计算的基础软件包），pandas（pandas是一种快速，强大，灵活且易于使用的开源数据分析和处理工具），matplotlib和seaborn绘图。

2.3.2.1 Step1：函数库导入

```
##  基础函数库
import numpy as np
import pandas as pd

## 绘图函数库
import matplotlib.pyplot as plt
import seaborn as sns
```

本次我们选择企鹅数据（palmerpenguins）进行方法的尝试训练，该数据集一共包含8个变量，其中7个特征变量，1个目标分类变量。共有150个样本，目标变量为 企鹅的类别 其都属于企鹅类的三个亚属，分别是(Adélie, Chinstrap and Gentoo)。包含的三种种企鹅的七个特征，分别是所在岛屿，嘴巴长度，嘴巴深度，脚蹼长度，身体体积，性别以及年龄。

变量	描述
species	a factor denoting penguin species
island	a factor denoting island in Palmer Archipelago, Antarctica
bill_length_mm	a number denoting bill length
bill_depth_mm	a number denoting bill depth
flipper_length_mm	an integer denoting flipper length
body_mass_g	an integer denoting body mass
sex	a factor denoting penguin sex
year	an integer denoting the study year

2.3.2.2 Step2：数据读取/载入

```
## 我们利用Pandas自带的read_csv函数读取并转化为DataFrame格式

data = pd.read_csv('penguins_raw.csv')
```

```
## 为了方便我们仅选取四个简单的特征，有兴趣的同学可以研究下其他特征的含义以及使用方法
data = data[['Species','Culmen Length (mm)','Culmen Depth (mm)',
             'Flipper Length (mm)','Body Mass (g)']]
```

2.3.2.3 Step3：数据信息简单查看

```
## 利用.info()查看数据的整体信息
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 5 columns):
Species                344 non-null object
Culmen Length (mm)     342 non-null float64
Culmen Depth (mm)      342 non-null float64
Flipper Length (mm)    342 non-null float64
Body Mass (g)          342 non-null float64
dtypes: float64(4), object(1)
memory usage: 13.6+ KB
```

```
## 进行简单的数据查看，我们可以利用 .head() 头部.tail()尾部
data.head()
```

	Species	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)
0	Adelie Penguin (Pygoscelis adeliae)	39.1	18.7	181.0	3750.0
1	Adelie Penguin (Pygoscelis adeliae)	39.5	17.4	186.0	3800.0
2	Adelie Penguin (Pygoscelis adeliae)	40.3	18.0	195.0	3250.0
3	Adelie Penguin (Pygoscelis adeliae)	NaN	NaN	NaN	NaN
4	Adelie Penguin (Pygoscelis adeliae)	36.7	19.3	193.0	3450.0

这里我们发现数据集中存在NaN，一般的我们认为NaN在数据集中代表了缺失值，可能是数据采集或处理时产生的一种错误。这里我们采用-1将缺失值进行填补，还有其他例如“中位数填补、平均数填补”的缺失值处理方法有兴趣的同学也可以尝试。

```
data = data.fillna(-1)
```

```
data.tail()
```

	Species	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)
339	Chinstrap penguin (Pygoscelis antarctica)	55.8	19.8	207.0	4000.0
340	Chinstrap penguin (Pygoscelis antarctica)	43.5	18.1	202.0	3400.0
341	Chinstrap penguin (Pygoscelis antarctica)	49.6	18.2	193.0	3775.0
342	Chinstrap penguin (Pygoscelis antarctica)	50.8	19.0	210.0	4100.0
343	Chinstrap penguin (Pygoscelis antarctica)	50.2	18.7	198.0	3775.0

```
## 其对应的类别标签为'Adelie Penguin', 'Gentoo penguin', 'Chinstrap penguin'三种不同企鹅的类别。
data['Species'].unique()
```

```
array(['Adelie Penguin (Pygoscelis adeliae)',
       'Gentoo penguin (Pygoscelis papua)',
       'Chinstrap penguin (Pygoscelis antarctica)'], dtype=object)
```

```
## 利用value_counts函数查看每个类别数量
pd.Series(data['Species']).value_counts()
```

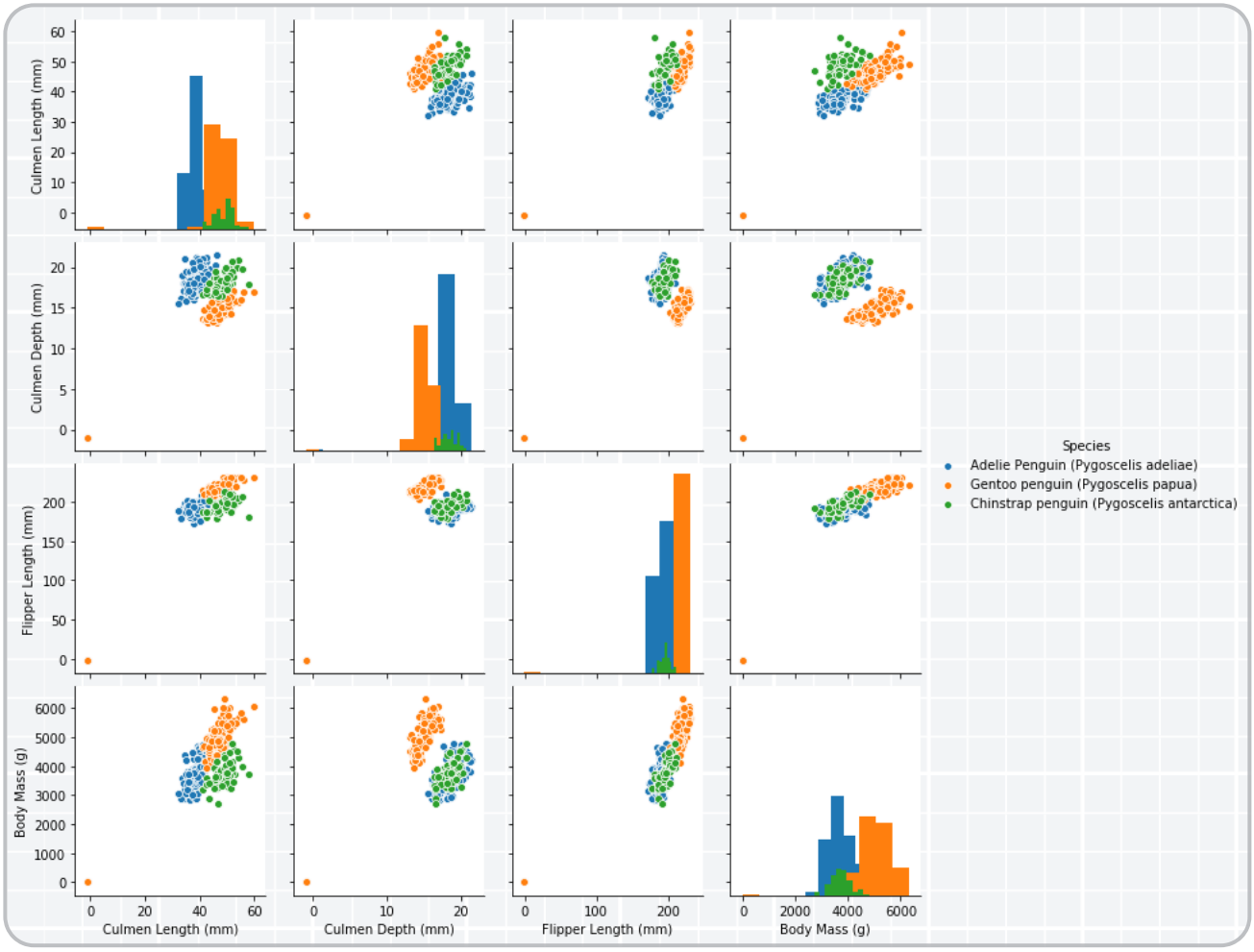
```
Adelie Penguin (Pygoscelis adeliae)          152
Gentoo penguin (Pygoscelis papua)            124
Chinstrap penguin (Pygoscelis antarctica)     68
Name: Species, dtype: int64
```

```
## 对于特征进行一些统计描述
data.describe()
```

	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)
count	344.000000	344.000000	344.000000	344.000000
mean	43.660756	17.045640	199.741279	4177.319767
std	6.428957	2.405614	20.806759	861.263227
min	-1.000000	-1.000000	-1.000000	-1.000000
25%	39.200000	15.500000	190.000000	3550.000000
50%	44.250000	17.300000	197.000000	4025.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

2.3.2.4 Step4：可视化描述

```
## 特征与标签组合的散点可视化
sns.pairplot(data=data, diag_kind='hist', hue= 'Species')
plt.show()
```

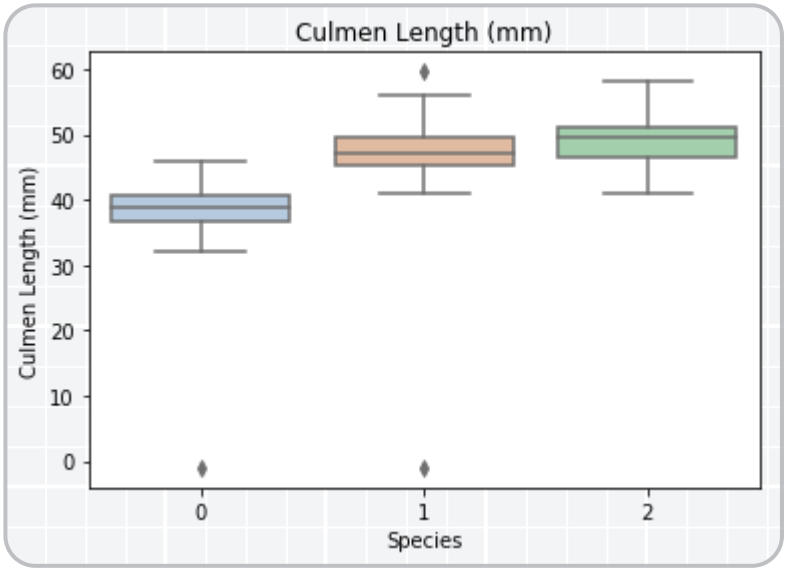


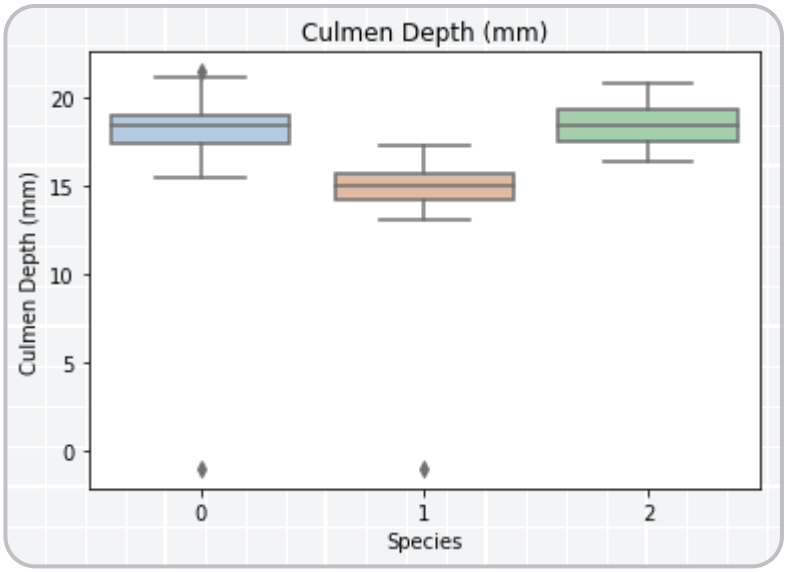
从上图可以发现，在2D情况下不同的特征组合对于不同类别的企鹅的散点分布，以及大概的区分能力。

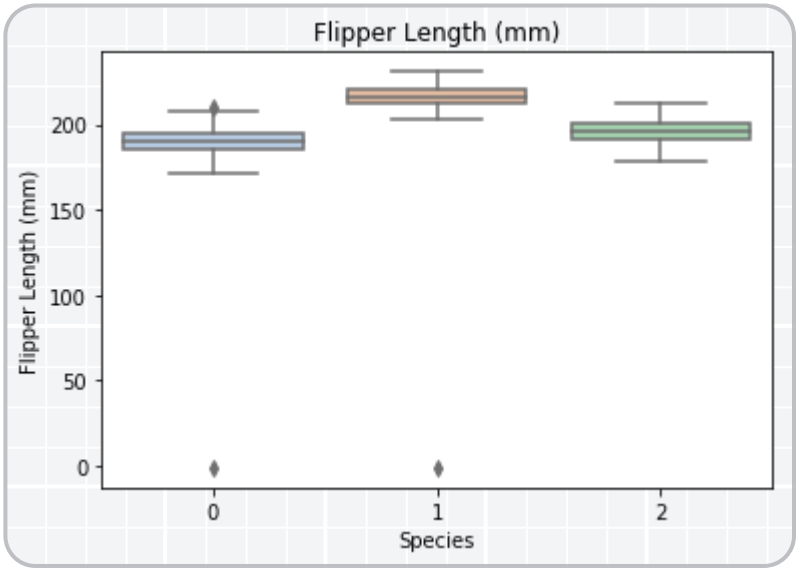
我们发现

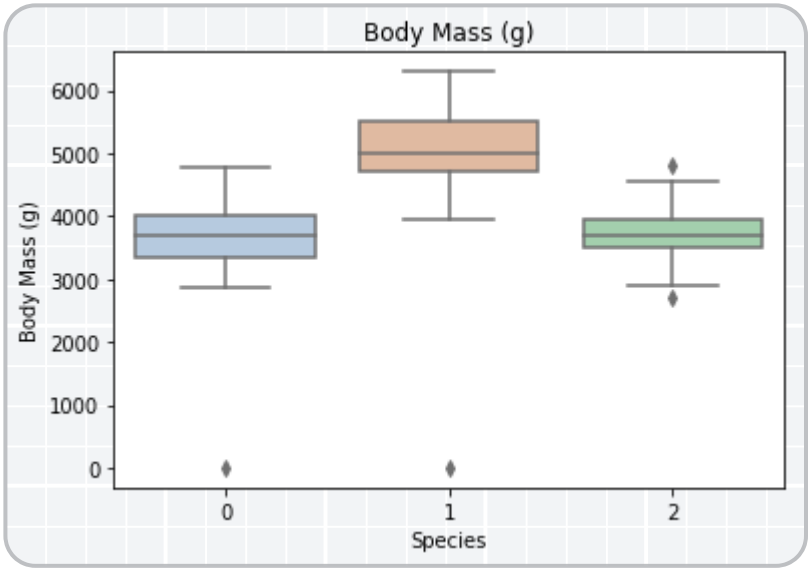
```
'''为了方便我们将标签转化为数字
       'Adelie Penguin (Pygoscelis adeliae)'        ------0
       'Gentoo penguin (Pygoscelis papua)'          ------1
       'Chinstrap penguin (Pygoscelis antarctica)   ------2 '''

def trans(x):
    if x == data['Species'].unique()[0]:
        return 0
    if x == data['Species'].unique()[1]:
        return 1
    if x == data['Species'].unique()[2]:
        return 2

data['Species'] = data['Species'].apply(trans)
```

```
for col in data.columns:
    if col != 'Species':
        sns.boxplot(x='Species', y=col, saturation=0.5, palette='pastel', data=data)
        plt.title(col)
        plt.show()
```


Culmen Length (mm)
Culmen Length (mm)
60
50
40
30
20
10
0
0
1
2
Species

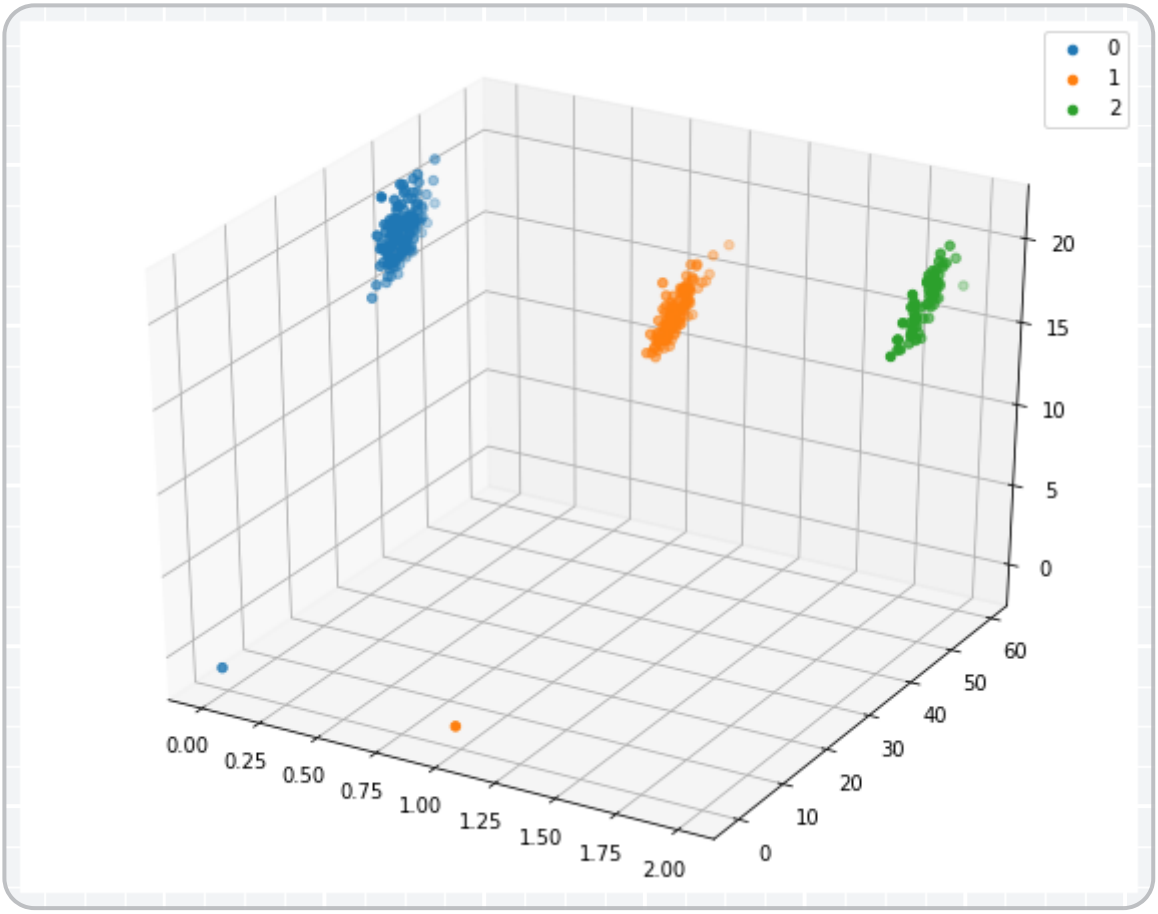

Culmen Depth (mm)
Culmen Depth (mm)
20
15
10
5
0
0
1
2
Species


Flipper Length (mm)
Flipper Length (mm)
200
150
100
50
0
0
1
2
Species



利用箱型图我们也可以得到不同类别在不同特征上的分布差异情况。

```
# 选取其前三个特征绘制三维散点图
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')

data_class0 = data[data['Species']==0].values
data_class1 = data[data['Species']==1].values
data_class2 = data[data['Species']==2].values
# 'setosa'(0), 'versicolor'(1), 'virginica'(2)
ax.scatter(data_class0[:,0], data_class0[:,1],
data_class0[:,2],label=data['Species'].unique()[0])
ax.scatter(data_class1[:,0], data_class1[:,1],
data_class1[:,2],label=data['Species'].unique()[1])
ax.scatter(data_class2[:,0], data_class2[:,1],
data_class2[:,2],label=data['Species'].unique()[2])
plt.legend()

plt.show()
```



2.3.2.5 Step5：利用决策树模型在二分类上进行训练和预测

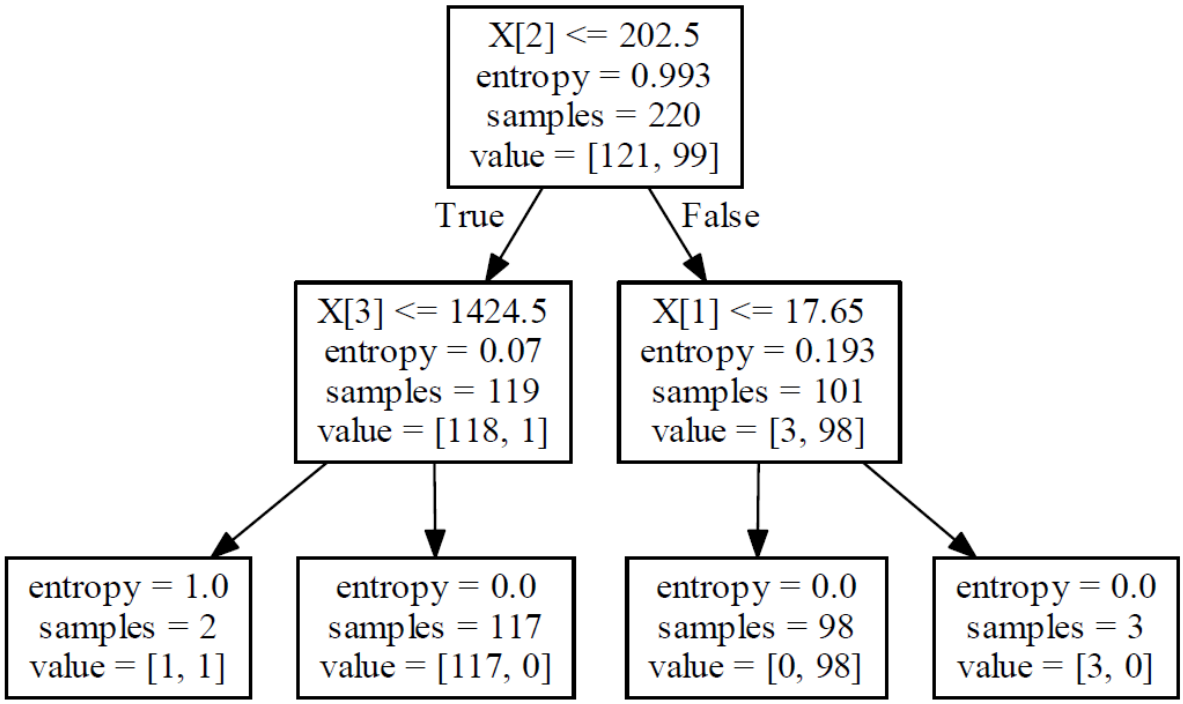
```
## 为了正确评估模型性能，将数据划分为训练集和测试集，并在训练集上训练模型，在测试集上验证模型性能。
from sklearn.model_selection import train_test_split

## 选择其类别为0和1的样本 （不包括类别为2的样本）
data_target_part = data[data['Species'].isin([0,1])][['Species']]
data_features_part = data[data['Species'].isin([0,1])][['Culmen Length (mm)','Culmen Depth (mm)',
            'Flipper Length (mm)','Body Mass (g)']]

## 测试集大小为20%， 80%/20%分
x_train, x_test, y_train, y_test = train_test_split(data_features_part, data_target_part, test_size = 0.2, random_state = 2020)
```

```
## 从sklearn中导入决策树模型
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
## 定义 逻辑回归模型
clf = DecisionTreeClassifier(criterion='entropy')
# 在训练集上训练决策树模型
clf.fit(x_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

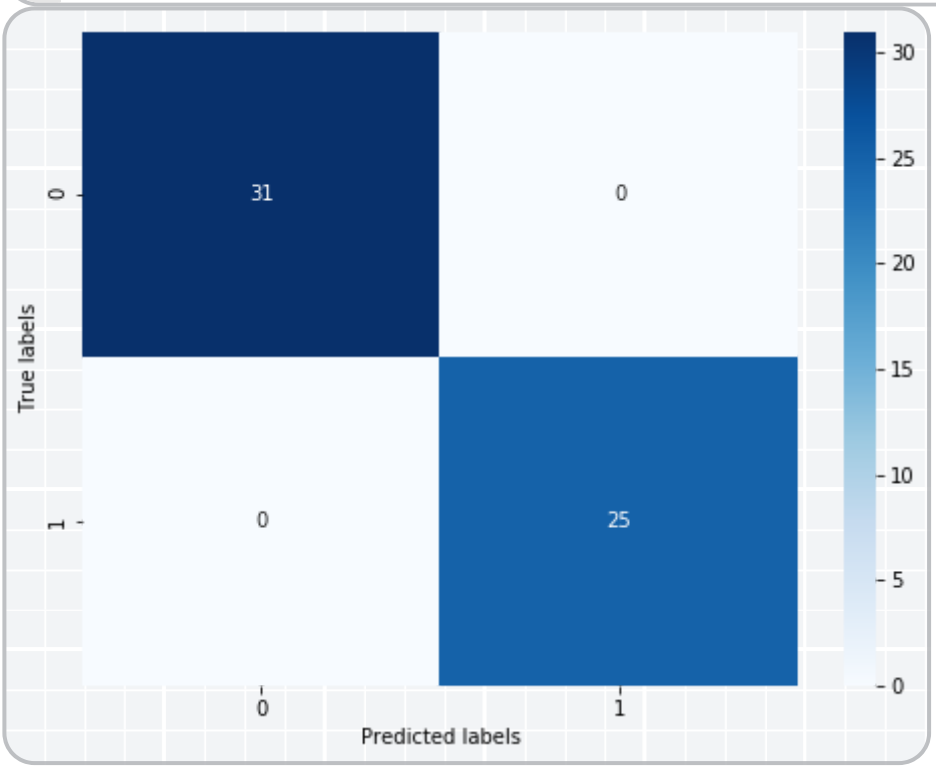
```
## 可视化
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("penguins")
```

```
'penguins.pdf'
```



```
## 在训练集和测试集上分布利用训练好的模型进行预测
train_predict = clf.predict(x_train)
test_predict = clf.predict(x_test)
from sklearn import metrics

## 利用accuracy（准确度）【预测正确的样本数目占总预测样本数目的比例】评估模型效果
print('The accuracy of the Logistic Regression is:',metrics.accuracy_score(y_train,train_predict))
print('The accuracy of the Logistic Regression is:',metrics.accuracy_score(y_test,test_predict))

## 查看混淆矩阵 (预测值和真实值的各类情况统计矩阵)
confusion_matrix_result = metrics.confusion_matrix(test_predict,y_test)
print('The confusion matrix result:\n',confusion_matrix_result)

# 利用热力图对于结果进行可视化
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_result, annot=True, cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
The accuracy of the Logistic Regression is: 0.9954545454545455
The accuracy of the Logistic Regression is: 1.0
The confusion matrix result:
 [[31  0]
 [ 0 25]]
```



我们可以发现其准确度为1，代表所有的样本都预测正确了。

2.3.2.6 Step6：利用逻辑回归模型在三分类(多分类)上进行训练和预测

```
## 测试集大小为20%，  80%/20%分
x_train, x_test, y_train, y_test = train_test_split(data[['Culmen Length (mm)','Culmen Depth (mm)',
            'Flipper Length (mm)','Body Mass (g)']], data[['Species']], test_size = 0.2, random_state = 2020)
## 定义 逻辑回归模型
clf = DecisionTreeClassifier()
# 在训练集上训练逻辑回归模型
clf.fit(x_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

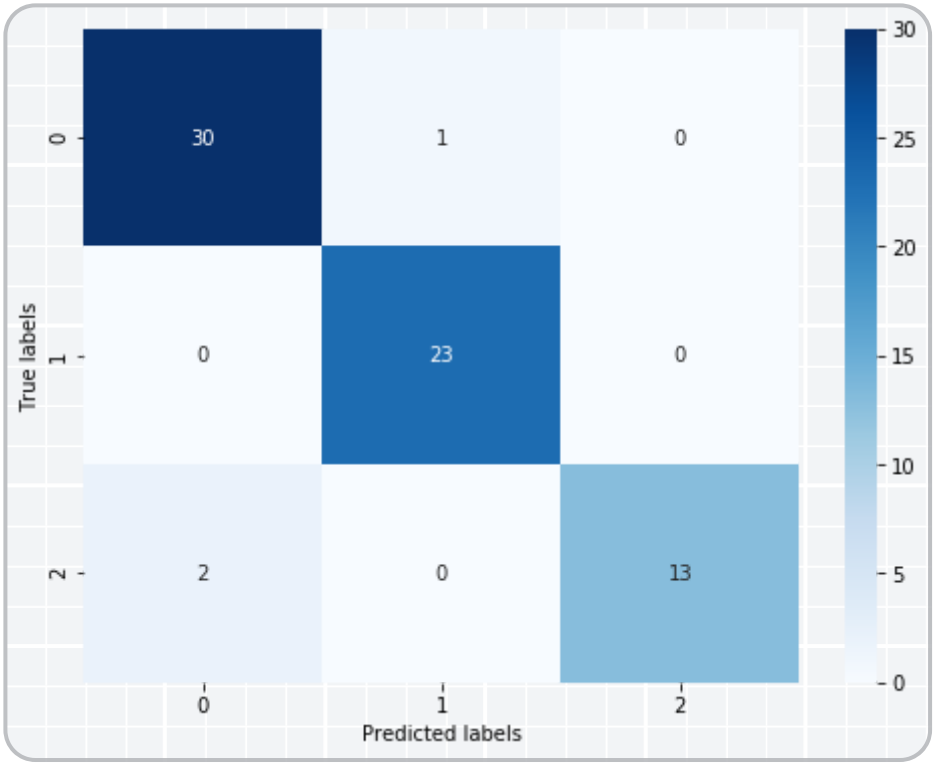
```
## 在训练集和测试集上分布利用训练好的模型进行预测
train_predict = clf.predict(x_train)
test_predict = clf.predict(x_test)

## 由于逻辑回归模型是概率预测模型（前文介绍的 p = p(y=1|x,\theta)）,所有我们可以利用 predict_proba 函数预测其概率
train_predict_proba = clf.predict_proba(x_train)
test_predict_proba = clf.predict_proba(x_test)

print('The test predict Probability of each class:\n',test_predict_proba)
## 其中第一列代表预测为0类的概率，第二列代表预测为1类的概率，第三列代表预测为2类的概率。

## 利用accuracy（准确度）【预测正确的样本数目占总预测样本数目的比例】评估模型效果
print('The accuracy of the Logistic Regression is:',metrics.accuracy_score(y_train,train_predict))
print('The accuracy of the Logistic Regression is:',metrics.accuracy_score(y_test,test_predict))
```

```
The test predict Probability of each class:
 [[0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
```

```
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]]
The accuracy of the Logistic Regression is: 0.9963636363636363
The accuracy of the Logistic Regression is: 0.9565217391304348
```

```python
## 查看混淆矩阵
confusion_matrix_result = metrics.confusion_matrix(test_predict,y_test)
print('The confusion matrix result:\n',confusion_matrix_result)

# 利用热力图对于结果进行可视化
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_result, annot=True, cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
The confusion matrix result:
 [[30  1  0]
 [ 0 23  0]
 [ 2  0 13]]
```



2.4 重要知识点

2.4.1 决策树构建的伪代码

输入： 训练集D={(x_1 , y_1),(x_2 , y_2),....,(x_m , y_m)};

特征集A={ a_1 , a_2 ,...., a_d }

输出： 以node为根节点的一颗决策树

过程：函数TreeGenerate(D , A)

1. 生成节点node
2. if D 中样本全书属于同一类别 C $then$ ：
3. ----将node标记为 C 类叶节点； $return$
4. if A = 空集 OR D中样本在 A 上的取值相同 $then$ ：
5. ----将node标记为叶节点，其类别标记为 D 中样本数最多的类； $return$
6. 从 A 中选择最优划分属性 a_* ；
7. for a_* 的每一个值 a_*^v do ：
8. ----为node生成一个分支，令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集；
9. ---- if D_v 为空 $then$ ：
10. --------将分支节点标记为叶节点，其类别标记为 D 中样本最多的类; $then$
11. ---- $else$ ：

12. --------以 TreeGenerate(D_v ，A ｛ a_* ｝)为分支节点

决策树的构建过程是一个递归过程。函数存在三种返回状态：（1）当前节点包含的样本全部属于同一类别，无需继续划分；（2）当前属性集为空或者所有样本在某个属性上的取值相同，无法继续划分；（3）当前节点包含的样本集合为空，无法划分。

2.4.2 划分选择

从上述伪代码中我们发现，决策树的关键在于line6.从 A 中选择最优划分属性 a_* ，一般我们希望决策树每次划分节点中包含的样本尽量属于同一类别，也就是节点的“纯度”更高。

2.4.2.1 信息增益

信息熵是一种衡量数据混乱程度的指标，信息熵越小，则数据的“纯度”越高

$$\mathrm{Ent}(D)=-\sum_{k=1}^{|\mathcal{Y}|}p_k\log_2 p_k$$

其中 p_k 代表了第 k 类样本在 D 中占有的比例。

假设离散属性 a 有 V 个可能的取值{ a^1 ，a^2 ,...., a^V }，若使用 a 对数据集 D 进行划分，则产生 D 个分支节点，记为 D^v 。则使用 a 对数据集进行划分所带来的信息增益被定义为：

$$\mathrm{Gain}(D,a)=\mathrm{Ent}(D)-\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v)$$

一般的信息增益越大，则意味着使用特征 a 来进行划分的效果越好。

2.4.2.2 基尼指数

$$\begin{aligned}\mathrm{Gini}(D)&=\sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k}p_k p_{k'}\\&=1-\sum_{k=1}^{|\mathcal{Y}|}p_k^2\end{aligned}$$

基尼指数反映了从数据集 D 中随机抽取两个的类别标记不一致的概率。

$$\text{Gini index}(D,a)=\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Gini}(D^v)$$

使用特征 a 对数据集 D 划分的基尼指数定义为上。

2.4.3 重要参数

2.4.3.1 criterion

Criterion这个参数正是用来决定模型特征选择的计算方法的。sklearn提供了两种选择：

1. 输入”entropy“，使用信息熵（Entropy）
2. 输入”gini“，使用基尼系数（Gini Impurity）

2.4.3.2 random_state & splitter

random_state用来设置分枝中的随机模式的参数，默认None，在高维度时随机性会表现更明显。splitter也是用来控制决策树中的随机选项的，有两种输入值，输入”best"，决策树在分枝时虽然随机，但是还是会优先选择更重要的特征进行分枝（重要性可以通过属性feature_importances_查看），输入“random"，决策树在分枝时会更加随机，树会因为含有更多的不必要信息而更深更大，并因这些不必要信息而降低对训练集的拟合。

2.4.3.3 max_depth

限制树的最大深度，超过设定深度的树枝全部剪掉。这是用得最广泛的剪枝参数，在高维度低样本量时非常有效。决策树多生长一层，对样本量的需求会增加一倍，所以限制树深度能够有效地限制过拟合。

2.4.3.4 min_samples_leaf

min_samples_leaf 限定，一个节点在分枝后的每个子节点都必须包含至少min_samples_leaf个训练样本，否则分枝就不会发生，或者，分枝会朝着满足每个子节点都包含min_samples_leaf个样本的方向去发生。一般搭配max_depth使用，在回归树中有神奇的效果，可以让模型变得更加平滑。这个参数的数量设置得太小会引起过拟合，设置得太大就会阻止模型学习数据。