第6课时 决策树与随机森林

主讲教师：欧新宇

January 19, 2020

Outlines

- 决策树的基本原理
- 决策树的构建
- 随机森林的基本原理
- 随机森林的构建
- 实例分析——基于Adult数据集的相亲问题

决策树的基本原理

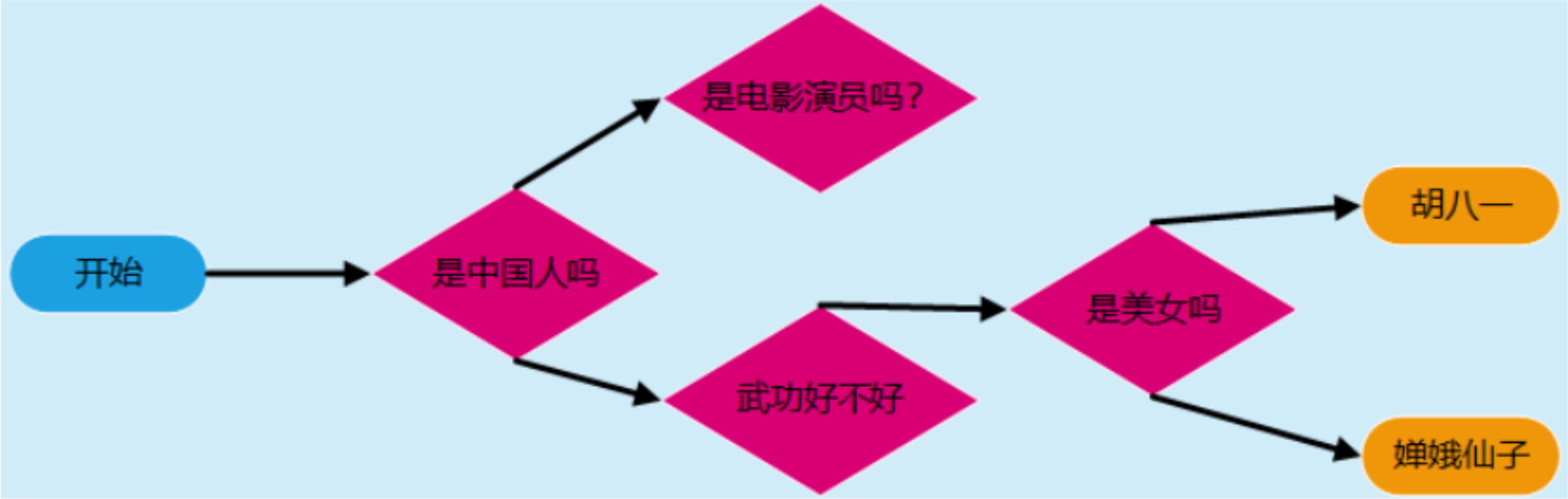
决策树(Decision Tree) 是一种基本的分类和回归算法（本课程仅讨论决策树在分类中的应用）。决策树模型呈树形结构，在分类问题中，依据样本的特征对实例进行分类，是一种典型的if-then/else-then推导规则。从数学原理上理解，它也可以被认为是定义在特征空间与类别空间上的**条件概率**分布。

- **学习时：** 利用训练数据，根据**损失函数最小化**的原则建立决策树模型;
- **预测时：** 对新的数据，利用决策树模型进行分类。

决策树学习通常包括3个步骤：特征选择、决策树的生成和决策树的修剪。

决策树的基本原理

下面给出一个利用决策树进行判定的图例，要求：根据一系列的属性特征输出特定的人物。



决策树的使用

载入数据库并进行数据预处理

[1]:
```
# 导入numpy计算库
import numpy as np

# 导入画图工具
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# 导入tree树模型和数据集加载工具
from sklearn import tree
from sklearn import datasets
# 导入数据拆分工具
from sklearn.model_selection import train_test_split
wine = datasets.load_wine()

# 设置X，y的值。此处为了便于可视化，仅选取前两个特征
X = wine.data[:, :2]
#X = wine.data
y = wine.target
# 将数据集拆分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

决策树的使用

配置决策树，并拟合训练集

[2]:
```
# 设置决策树的分类器的最大深度为 1
clf = tree.DecisionTreeClassifier(max_depth = 1)
# 拟合训练数据集
clf.fit(X_train, y_train)
```

[2]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=1,
                       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')
```

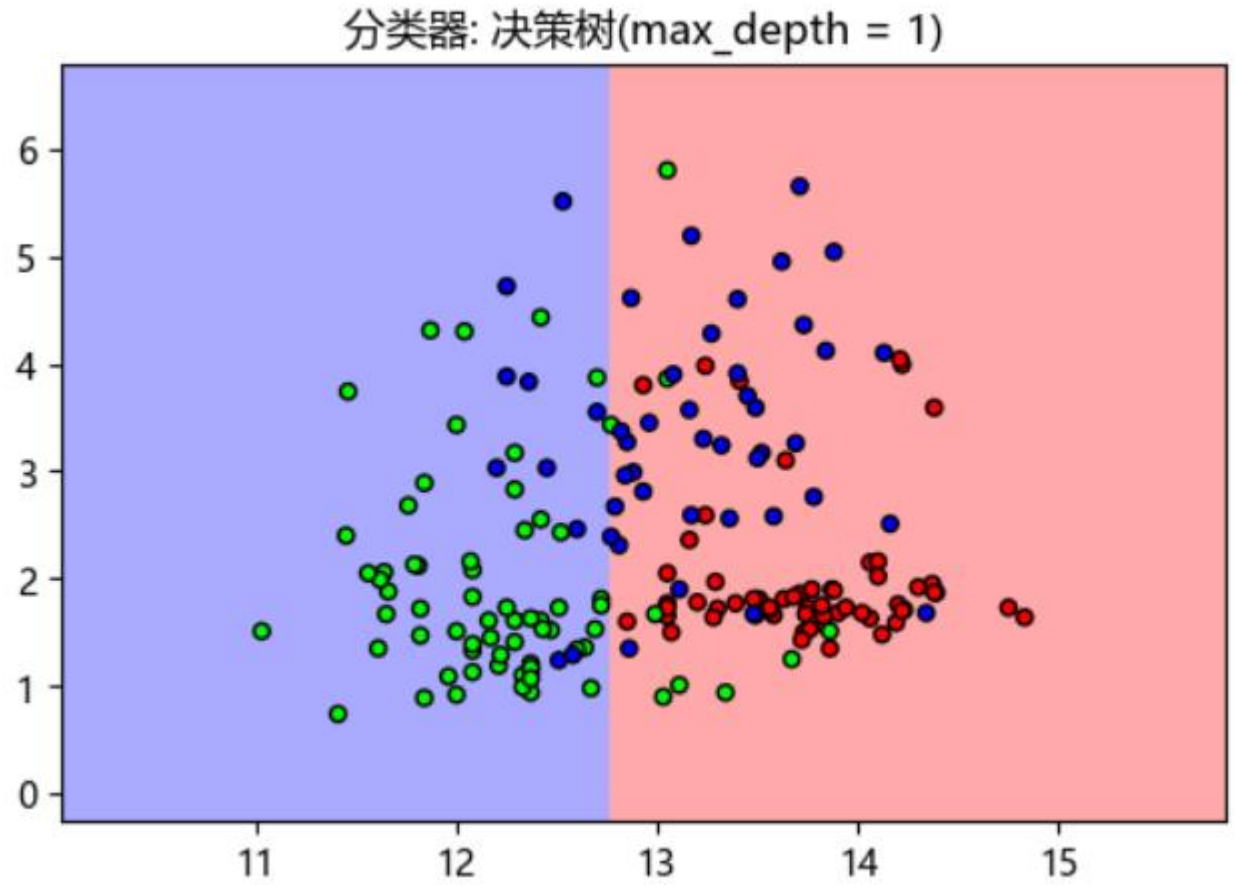
决策树的使用

可视化分类器结果

[3]:
```
# 定义图像中分区的颜色和散点的颜色
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

# 分别用样本的两个特征值创建图像的横轴和纵轴
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
# 设置特征轴的尺度的粒度
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# 给每个分类中的样本分配不同的颜色
Z = Z.reshape(xx.shape)
plt.figure(dpi = 100)
plt.rcParams['font.sans-serif'] = [u'Microsoft YaHei']
plt.pcolormesh(xx, yy, Z, cmap = cmap_light)


# 用散点把样本表示出来
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = cmap_bold, edgecolor = 'k', s = 20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("分类器: 决策树(max_depth = 1)")

plt.show()
```

决策树的使用

- 可视化分类器结果



决策树的使用

对比不同树深模型在训练集和测试集上的准确率

[9]:
```
# 输出模型评分，即正确率
score_train = clf.score(X_train, y_train)
score3_train = clf3.score(X_train, y_train)
score5_train = clf5.score(X_train, y_train)
score_test = clf.score(X_test, y_test)
score3_test = clf3.score(X_test, y_test)
score5_test = clf5.score(X_test, y_test)

print("模型一(树深 = 1)：训练集准确率：{0:.3f}，测试集准确率：{1:.3f}".format(score_train, score_test))
print("模型二(树深 = 3)：训练集准确率：{0:.3f}，测试集准确率：{1:.3f}".format(score3_train, score3_test))
print("模型三(树深 = 5)：训练集准确率：{0:.3f}，测试集准确率：{1:.3f}".format(score5_train, score5_test))
```

```
模型一(树深 = 1)：训练集准确率：0.669，测试集准确率：0.689
模型二(树深 = 3)：训练集准确率：0.850，测试集准确率：0.933
模型三(树深 = 5)：训练集准确率：0.887，测试集准确率：0.911
```

由结果可以得到以下结论：

1. 随着决策树的深度增加，模型的性能得到了一定的提升（*在一定范围内*）；
2. 深度增加到5时，模型出现了过拟合现象。即：训练集性能持续提高，测试集性能出现了降低。

决策树的优缺点

- **优点：**

  计算复杂度低，分类速度快，模型易于理解、可读性高，对中间值的缺失不敏感

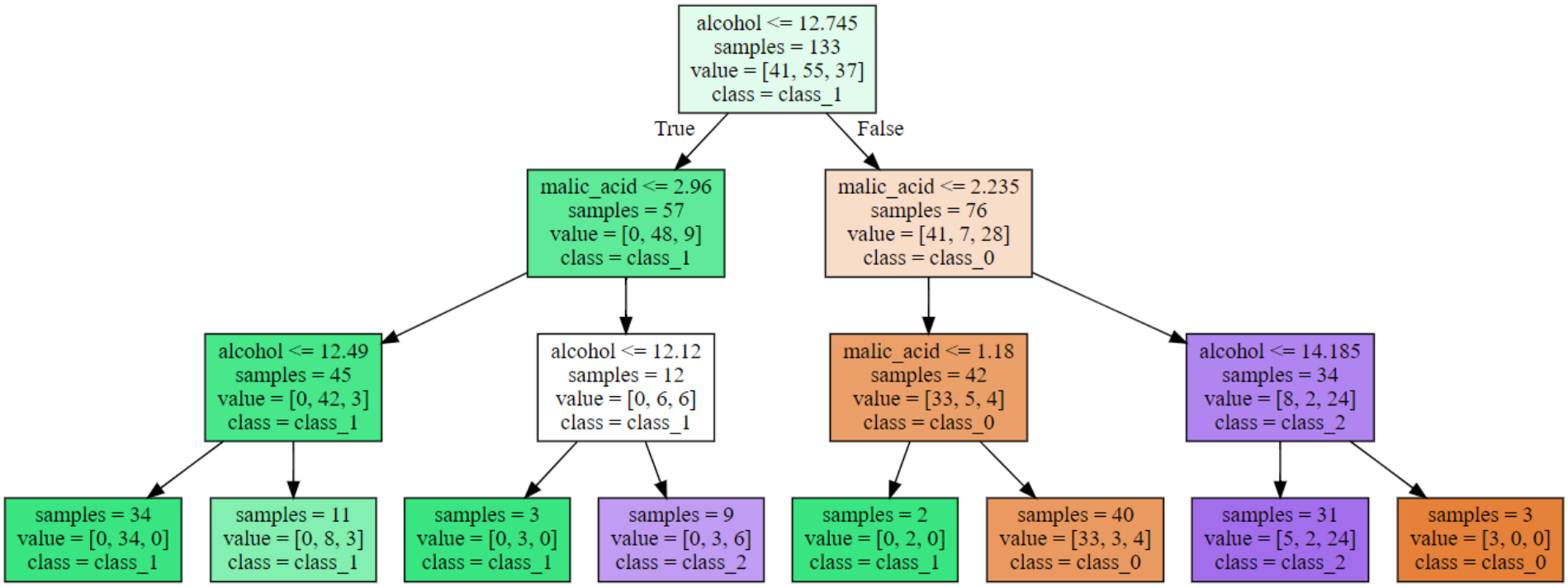
- **缺点：**

  可能会产生过度匹配问题

- **适用数据类型：**数值型和标称型

决策树的工作过程的可视化

Graphviz (Graph Visualization Software) 是一个由AT&T实验室启动的开源工具包，可用于绘制各种流程图和结构图。它依赖于DOT描述语言，DOT是一种图形描述语言，具有简单易学的特点。



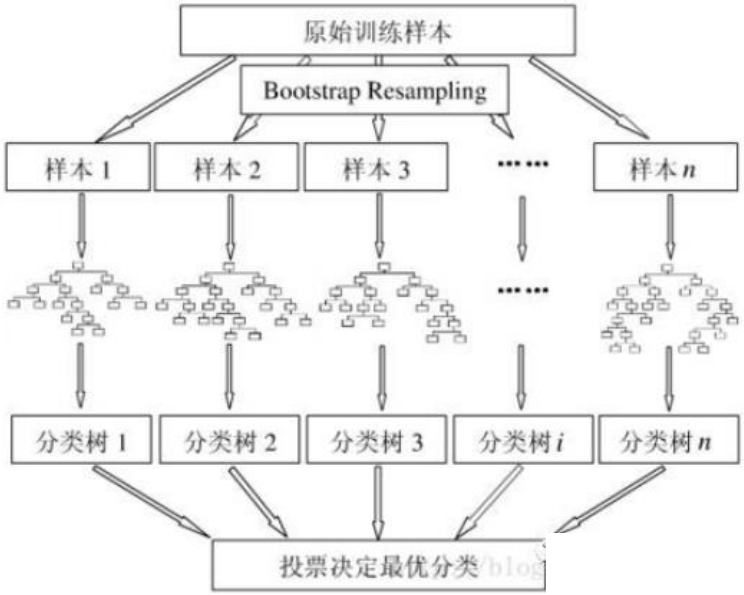
随机森林的基本概念

在机器学习中，**随机森林**是一个包含多个决策树的分类器，其输出的类别是由每棵树的输出进行联合判决。随机森林最初是由Leo Breiman和Adele Cutler提出，并以"**Random Forests**"作为商标，而该术语则是由贝尔实验室的Tin Kam Ho所提出的随机决策森林（random decision forests）而来。

随机森林的基本概念

随机森林算法过程

1. 假设用 N 来表示训练样本的个数，M 表示特征的数目；
2. 输入特征数目 m，用于确定决策树上一个节点的决策结果；其中 m 应远小于 M；
3. 从 N 个训练样本中以（放回）抽样的方式，取样 N 次，形成一个训练集（即 bootstrap取样），并用未抽到的用例（样本）作预测，评估其误差；
4. 对于每一个节点，随机选择 m个特征，决策树上每个节点的决定都是基于这些特征确定的。根据这 m 个特征，计算其最佳的分裂方式。
5. 每棵树都会完整成长而不会剪枝，这有可能在建完一棵正常树状分类器后会被采用。



随机森林的使用

数据预载入及预处理

[10]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split

# 导入随机森林模型
from sklearn.ensemble import RandomForestClassifier

wine = datasets.load_wine()
X = wine.data[:, :2] # 为便于可视化仍然仅使用前两个特征
y = wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

随机森林的使用

模型训练及常用参数

[11]:
```
# 设定随机森林中树的数量，此处 = 6
forest = RandomForestClassifier(n_estimators = 6, random_state = 3, n_jobs = -1)
forest.fit(X_train, y_train)
```

[11]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', 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, n_estimators=6, n_jobs=-1,
                       oob_score=False, random_state=3, verbose=0,
                       warm_start=False)
```

- ✓ **Bootstrap**
- ✓ **Class_weight**
- ✓ **Max_feature**
- ✓ **Max_leaf_nodes**
- ✓ **n_estimators**
- ✓ **n_jobs**
- ✓ **random_state**

随机森林的使用

模型评估

[12]:
```
score_train = forest.score(X_train, y_train)
score_test = forest.score(X_test, y_test)

print("随机森林：\n 训练集准确率：{0:.3f}，测试集准确率：{1:.3f}".format(score_train, score_test))
```

```
随机森林：
 训练集准确率：0.955，测试集准确率：0.778
```

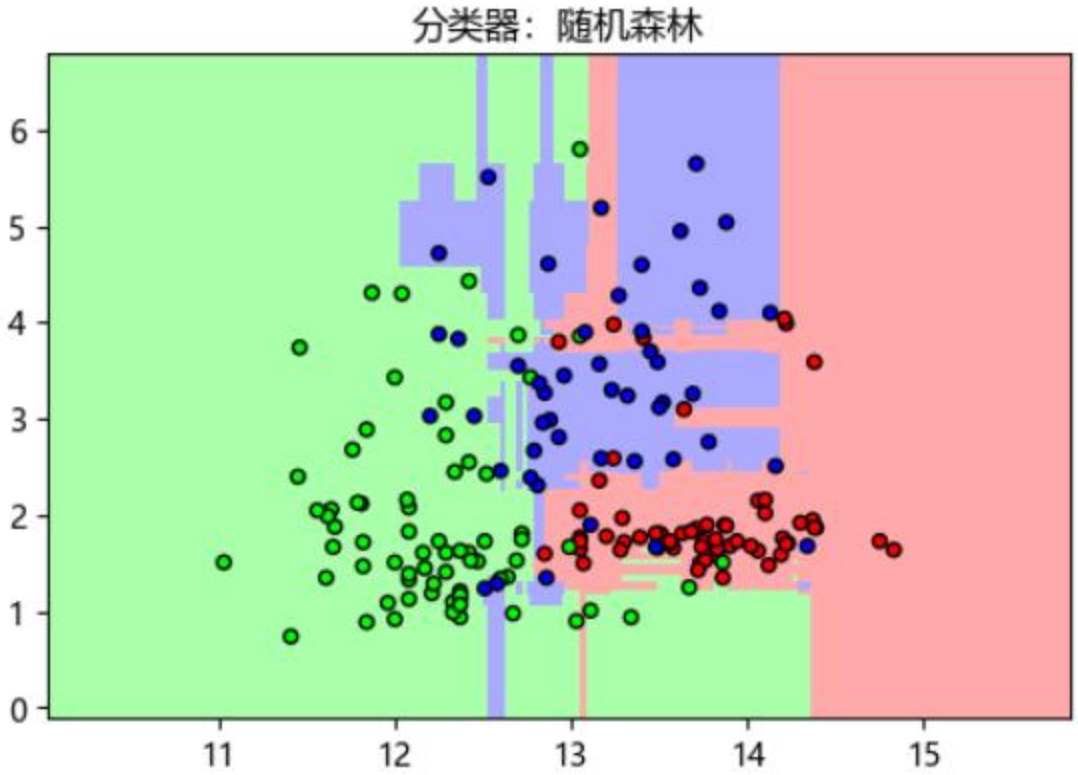
随机森林的使用

结果可视化

[13]:
```
# 定义图像中分区的颜色和散点的颜色
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

# 分别用样本的两个特征值创建图像的横轴和纵轴
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
# 设置特征轴的尺度的粒度
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

Z = forest.predict(np.c_[xx.ravel(), yy.ravel()])

# 给每个分类中的样本分配不同的颜色
Z = Z.reshape(xx.shape)
plt.figure(dpi = 100)
plt.rcParams['font.sans-serif'] = [u'Microsoft YaHei']
plt.pcolormesh(xx, yy, Z, cmap = cmap_light)

# 用散点把样本表示出来
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = cmap_bold, edgecolor = 'k', s = 20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("分类器：随机森林")

plt.show()
```

随机森林的使用

- 结果可视化



从结果看，随机森林获得的结果要更加细腻。有兴趣的同学，可以尝试调整一下参数n_estimator和random_state，看看是否能获得更好的预测结果。

随机森林的优缺点

- **优点**
  - 由于采用了集成算法，本身精度比大多数单个算法要好，所以准确性高
  - 在测试集上表现良好，由于两个随机性的引入，使得随机森林不容易陷入过拟合（样本随机，特征随机）
  - 在工业上，由于两个随机性的引入，使得随机森林具有一定的抗噪声能力，对比其他算法具有一定优势
  - 由于树的组合，使得随机森林可以处理非线性数据，本身属于非线性分类（拟合）模型
  - 它能够处理很高维度（feature很多）的数据，并且不用做特征选择，对数据集的适应能力强：既能处理离散型数据，也能处理连续型数据，数据集无需规范化
  - 训练速度快，可以运用在大规模数据集上

随机森林的优缺点

- **优点**
  - 在训练过程中，能够检测到feature间的互相影响，且可以得出feature的重要性，具有一定参考意义
  - 在训练过程中，能够检测到feature间的互相影响，且可以得出feature的重要性，具有一定参考意义
  - 由于每棵树可以独立、同时生成，容易做成并行化方法
  - 由于实现简单、精度高、抗过拟合能力强，当面对非线性数据时，适于作为基准模型（Baseline）
- **缺点**
  - 当随机森林中的决策树个数很多时，训练时需要的空间和时间会比较大
  - 随机森林中还有许多不好解释的地方，有点算是黑盒模型
  - 在某些噪音比较大的样本集上，模型容易陷入过拟合

实例分析——基于Adult数据集的相亲问题

- 载入数据集（Ch0604CaseAdult.py）
- 数据预处理
- 基于训练集进行建模
- 预测与评分
- 模型优化 (Ch0604CaseAdultFull.py)
- 超参数分析 (Ch0604CaseAdultFullDraw.py)

欧老师的联系方式

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Tel：18687840023