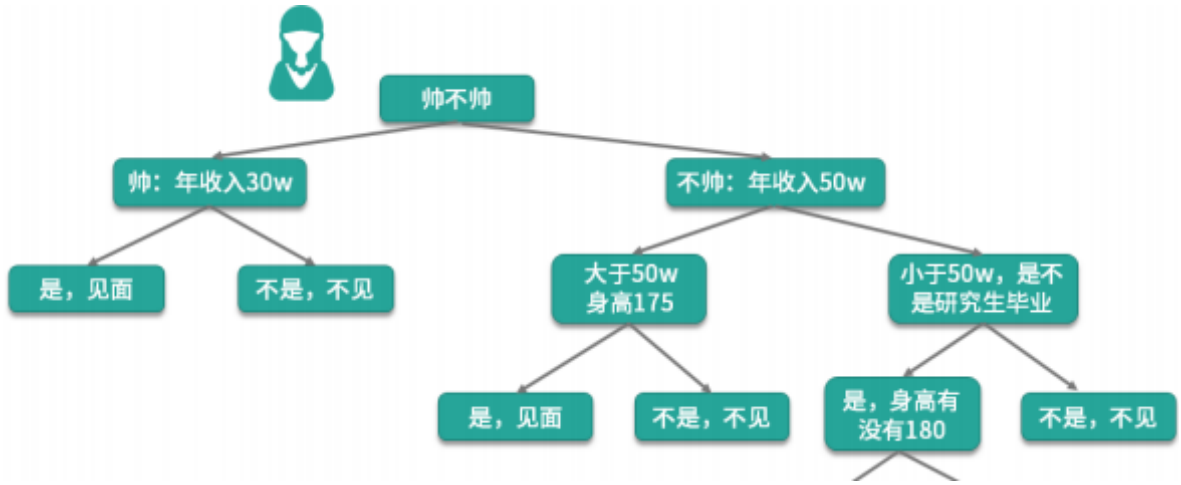
今天是分类算法的第二课时，我们今天要介绍的是一个应用非常广泛的模型——**决策树**。首先我依然会从一个例子出发，看看女神是怎样决策要不要约会的；然后分析它的算法原理、思路形成的过程；由于决策树非常有价值，还衍生出了很多高级版本，在扩展内容里我也进行了简要的介绍。希望通过本课时的学习，你可以掌握决策树的思路以及使用方法，并能够尝试用它来解决遇到的问题。



一个例子

我们都知道女神身后有很多的追求者，她肯定不会和每个人都约会，因为时间不够，必须要好好管理自己的时间才行。于是女神给每个想要约会的人发信息说：“把你的简历发过来吧。”

简历收上来后，第一眼先看照片，颜值打几分？然后再看年收入，长得帅的就可以少挣点，毕竟“帅也可以当饭吃啊”。不帅的呢？那收入必须要求高一点，“颜值不够，薪资来凑”。薪资还差点的，再看看学历是不是研究生 / 985/211，看看身高有没有 180…… 所以你就可以对号入座了，发现自己哪条都不符合，好了，去好好“搬砖” 吧。

由此可知，女神的筛选条件有颜值、身高、收入、学历等，每一项都会对最后是否约会的结果产生影响，即女神通过对这几种条件的判断，决定是否要安排约会。

上面这个过程就是决策树的思路，下面我们来看一下决策树的具体原理。

算法原理

在已知的条件中，选取一个条件作为树根，也就是作为第一个决策条件，比如“颜值” 分为帅和不帅两个结果，然后再看是否还需要其他判断条件。如果需要的话，继续构建一个分支来判断第二个条件，以此类推，直到能够推出一个结果，这个分支就结束了。

同样的，当我们把所有样本数据中出现的情况组合都构建入这棵树的时候，我们的算法也就完成了对样本的学习。最终形成的这棵树上，所有的叶子节点都是要输出的类别信息，所有的非叶子节点都是特征信息。当一个新的数据来了之后，就按照对应的判断条件，从根节点走到叶子节点，从而获得这个数据的分类结果。

比如，我帮女神收集了几份简历，然后按照条件整理出如下结果：

编号	颜值	年收入	身高	学历
1	帅	28 w	178 cm	本科
2	不帅	100 w	176 cm	硕士
3	不帅	40 w	185 cm	硕士

根据女神已经制定好的决策树，我们去预测一下这三个人能否获得约会资格。第一个人帅，那就从根节点走向左分支；再判断第二个条件：年收入只有 28w，不到 30w 的标准，那就被淘汰了。第二个人不帅，走向右分支，年收入 100w，那就可以继续往下进入左分支，身高 176cm 刚好过合格线，获得左分支的结果，顺利进入约会环节。第三个人不帅，年收入 40w，那就进入右分支，还需要再看学历：是研究生，那还可以继续走向左分支，身高有 185cm，那也成功获得约会资格。

这就是决策树最初的一个思路。但是这里有一个问题，我想你可能也会想到，那就是该如何选择一个特征作为根节点？下一次决策又该选取哪个特征作为节点？决策树算法使用了一种称作**信息增益**的方法来衡量一个特征和特征之间的重要性，信息增益越大表明这个特征越重要，那么就优先对这个特征进行决策。至于信息增益和信息熵是在信息论中涉及的内容，如果你有兴趣可以再进行详细学习。

在一种理想的情况下，我们构建的决策树上的每一个叶子节点都是一个纯粹的分类，也就是通过这条路径进入到这个叶子节点的所有数据都是同一种类别，但是这需要反复回溯修改非叶子节点的判定条件，而且要划分更多的分支来进行处理，所以实际上决策树实现的时候都采用了**贪心算法**，来寻找一个最近的最优解，而不是全局的最优解。

算法的优缺点

几个版本的决策树的比较

决策树最初的版本称为 ID3（Iterative Dichotomiser 3），ID3 的缺点是无法处理数据是连续值的情况，也无法处理数据存在缺失的问题，需要在准备数据环节把缺失字段进行补齐或者删除数据。后来有人提出了改进方案称为 C4.5，加入了对连续值属性的处理，同时也可以处理数据缺失的情况。同时，还有一种目前应用最多的 CART（Classification And Regression Tree）分类与回归树，每次分支只使用二叉树划分，同时可以用于解决回归问题。

关于这三种决策树，我列了一个对比的表格，可以看到它们之间的区别。

决策树	模型类型	树结构	特征选择	连续值处理	缺失值处
ID3 （Iterative Dichotomiser 3）	分类	多叉树	信息增益	不可以	不可以
C4.5	分类	多叉树	信息增益比	可以	可以
CART （Classification And Regression Tree）	分类与回归	二叉树	基尼系数	可以	可以

这里的优缺点是针对 CART 树来讲，因为现在 CART 是主流的决策树算法，而且在 sklearn 工具包中使用的也是 CART 决策树。那么我们再看一下，决策树算法有什么优缺点。

优点

- **非常直观，可解释极强。** 在生成的决策树上，每个节点都有明确的判断分支条件，所以非常容易看到为什么要这样处理，比起神经网络模型的黑盒处理，高解释性的模型非常受金融保险行业的欢迎。在后面的动手环节，我们能看到训练完成的决策树可以直接输出出来，以图形化的方式展示给我们生成的决策树每一个节点的判断条件是什么样子的。
- **预测速度比较快。** 由于最终生成的模型是一个树形结构，对于一条新数据的预测，只需要按照条件在每一个节点进行判定就可以。通常来说，树形结构都有助于提升运算速度。
- **既可以处理离散值也可以处理连续值，还可以处理缺失值。**

缺点

- **容易过拟合。** 试想在极端的情况下，我们根据样本生成了一个最完美的树，那么样本中出现的每一个值都会有一条路径来拟合，所以如果样本中存在一些问题数据，或者样本与测试数据存在一定的差距时，就会看出泛化性能不好，出现了过拟合的现象。
- **需要处理样本不均衡的问题。** 如果样本不均衡，某些特征的样本比例过大，最终的模型结果将会更偏向这些特征。
- **样本的变化会引发树结构巨变。**

关于剪枝

上面提到的一个问题就是决策树容易过拟合，那么我们需要使用剪枝的方式来使得模型的泛化能力更好，所以剪枝可以理解为简化我们的决策树，去掉不必要的节点路径以提高泛化能力。剪枝的方法主要有**预剪枝**和**后剪枝**两种方式。

- **预剪枝：** 在决策树构建之初就设定一个阈值，当分裂节点的熵阈值小于设定值的时候就不再进行分裂了；然而这种方法的实际效果并不是很好，因为谁也没办法预料到我们设定的恰好是我们想要的。
- **后剪枝：** 后剪枝方法就是在我们的决策树已经构建完成以后，再根据设定的条件来判断是否要合并一些中间节点，使用叶子节点来代替。在实际的情况下，通常都是采用后剪枝的方案。

尝试动手

关于几个版本的决策树你已经有了大致的了解，那么下面我们来动手写一写使用决策树算法进行的代码。

在前面的部分我没有写更多的注释，这部分是关于包的引入和导入数据的，在前面的章节已经进行过介绍，这里主要是把引入的算法包进行了调整，其余的部分实际上没有什么修改，如果你忘了可以看一下前面的章节。

```
from sklearn import datasets


from sklearn.tree import DecisionTreeClassifier


import numpy as np


np.random.seed(0)


iris=datasets.load_iris()


iris_x=iris.data


iris_y=iris.target


indices = np.random.permutation(len(iris_x))


iris_x_train = iris_x[indices[:-10]]

```

```
iris_y_train = iris_y[indices[:-10]]

iris_x_test  = iris_x[indices[-10:]]

iris_y_test  = iris_y[indices[-10:]]
```

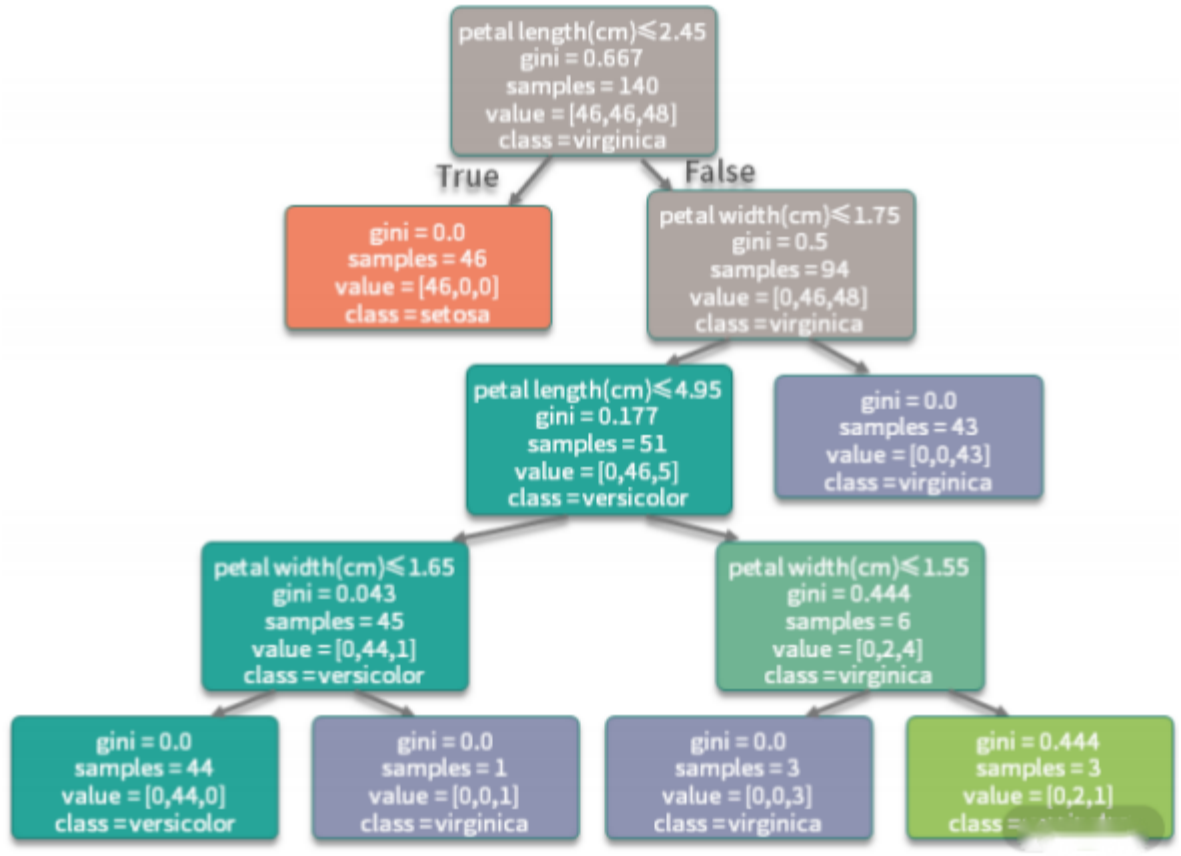
在模型训练时，我们设置了树的最大深度为 4。

```
clf = DecisionTreeClassifier(max_depth=4)

clf.fit(iris_x_train, iris_y_train)
```

根据上面的介绍，我们可以知道，经过调用 fit 方法进行模型训练，决策树算法会生成一个树形的判定模型，今天我们尝试把决策树算法生成的模型使用画图的方式展示出来。

```
from IPython.display import Image

from sklearn import tree

import pydotplus

dot_data = tree.export_graphviz(clf, out_file=None,

feature_names=iris.feature_names,

class_names=iris.target_names,

filled=True, rounded=True,

special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data)

Image(graph.create_png())
```

经过运行上面的代码，就会输出下面这幅图，可以看到每一次的判定条件以及基尼系数，还有能够落入此决策的样本数量和分类的类别。



看完了模型，我们使用模型来对测试数据进行一下预测。这里调用的预测方法跟之前都是一样的，这里就不详细介绍了。

```
iris_y_predict = clf.predict(iris_x_test)


score=clf.score(iris_x_test,iris_y_test,sample_weight=None)


print('iris_y_predict = ')


print(iris_y_predict)


print('iris_y_test = ')


print(iris_y_test)


print('Accuracy:',score)
```

我们看一下输出的结果，可以看到第二个测试样本预测错误了，其他的都预测正确，准确率是 90%。

```
iris_y_predict =


[1 2 1 0 0 0 2 1 2 0]
```

```
iris_y_test =


[1 1 1 0 0 0 2 1 2 0]


Accuracy: 0.9
```

今天我们的代码主要实现了工具包中的决策树算法，同时新增了一块输出决策树模型的方法，你是否也想亲手来试一下呢？下面我再讲解一些扩展内容，看看决策树算法在后续的演进中都有什么样的变化。

扩展内容

- 随机森林：为了更好地解决泛化及树结构变动~等~问题，从决策树演进出来随机森林算法。根据我们前面讲的模型集成方法，随机森林就是使用了 bagging 方案构建了多棵决策树，然后对所有树的结果来进行平均计算以获得最终的结果。
- GBDT：在随机森林的基础上，研究者又提出了梯度提升决策树算法（Gradient Boosting Decision Tree，GBDT），GBDT 是基于 boosting 的策略。与随机森林一样的是，GBDT 也会构建多棵决策树；但不同的是，GBDT 构建的多棵树之间是有联系的，每个分类器在上一轮分类器的**残差**基础上进行训练。
- XGBoost：一个非常火热的模型，有 “机器学习大杀器” 之称，在很多比赛中都获得了非常好的结果。但实际上 XGBoost 不算是一个算法，而是对 GBDT 的一种工程实现，它优化了 GBDT 里面的求解过程，并加入了很多工程上的优化项目，使得数据处理、运算速度等环节都有了很大的提升。

总结

这一小节的课程，我们讲解了第二个分类算法——决策树算法，首先从女神如何决策跟谁约会的问题出发，引出了决策树算法的原理，由于决策树算法非常容易理解，效果很好而且易于解释，所以研究者提出了各种各样的改进方案，并由决策树延伸出了很多新的优秀的算法。在尝试动手的环节，本课时加入了一些绘图的技巧，希望大家也能够掌握这部分的内容，这样也可以提升工作效率。

看完了决策树算法的介绍，不知道你是否对其中的细节还有什么疑惑？你可以在评论中写下来与大家一起讨论。