

棕榈学院

7 天 Python 进阶训练营讲义

第五讲



2019 年 1 月 14 日-2019 年 1 月 20 日

目录

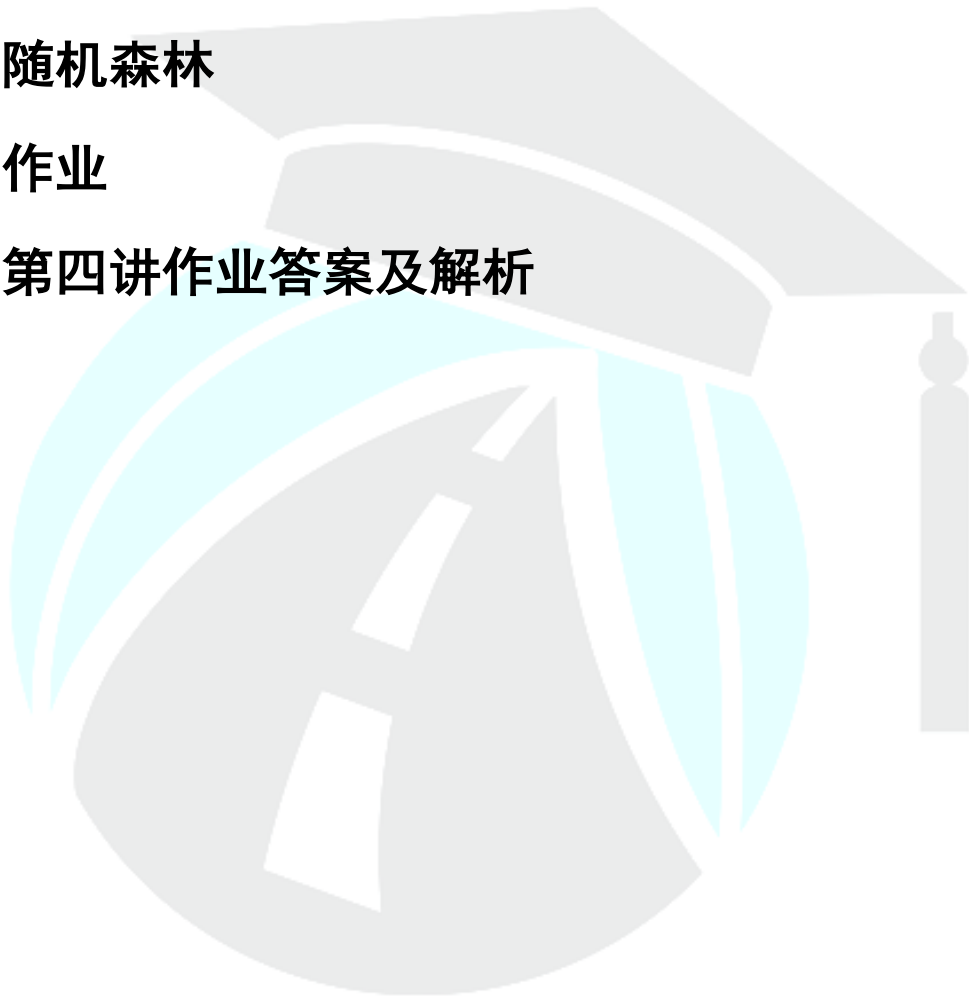
一、逻辑回归

二、决策树

三、随机森林

四、作业

五、第四讲作业答案及解析



在前几节课我们已经对模型进行了训练并学习了交叉验证,并对逻辑回归、决策树、随机森林三个模型的结果也有了大概的了解,接下来我们要更加深入去理解这三个不一样的模型,对其中的参数进行讲解。

首先看一下数据及变量

```
In [170]: #观察一下数据, 这个函数可以供你看到除了表头的前五行数据
xTrain.head()

Out[170]:
```

	Pclass	Age	SibSp	Parch	Fare	Title	FamilySize	IsAlone	Sex_Code	Embarked_Code	AgeBin_Code	FareBin_Code
140	3	19.0	0	2	15.2458	3	3	0	0	0	1	2
439	2	31.0	0	0	10.5000	1	1	1	1	1	2	1
817	2	31.0	1	1	37.0042	1	3	0	1	0	1	3
378	3	20.0	0	0	4.0125	1	1	1	1	0	1	0
491	3	21.0	0	0	7.2500	1	1	1	1	2	1	0

查看训练集的数据维度。

```
In [171]: ##查看训练集的数据维度
xTrain.shape
```

```
Out[171]: (712, 12)
```

要讲解三个不同模型, 所以在这里分别创建这三个对象。

```
In [172]: #分别创建名为logr, dtree, rf的逻辑回归, 决策树和随机森林对象
logr = LogisticRegression()
dtree = DecisionTreeClassifier()
rf = RandomForestClassifier()
```

用了最基本模型中的参数, 没有经过任何调整。

一、逻辑回归

1. 先让模型 fit x 和 y。

```
In [173]: #调用LogisticRegression的fit方法对数据集进行拟合, C为正则化系数λ的倒数, 通常默认为1, class_weight参数用于标示分类模型中各种类型的权重,
#dual: 一个布尔值, 如果为true, 则求解对偶形式(只在penalty='l2'且solver='lib-linear'有对偶形式), 如果为false, 则求解原始形式,
#默认为false; fit_intercept, 是否存在截距, 默认存在; intercept_scaling: 一个浮点数, 只有当solver='liblinear'才有意义。当采用
#fit_intercept时相当于人造一个特征出来, 特征恒为1, 权重为b。在计算正则化项的时候, 该人造特征也被考虑了, 因此为了降低这个人造特征的影响,
#需要提供intercept_scaling; max_iter: 一个整数, 指定最大迭代次数; multi_class: 一个字符串, 指定对于多分类问题的策略, 可以为如下值,
#ovr: 采用one-vs-rest策略; multinomial: 直接采用多分类逻辑回归策略。n_jobs: 一个正数, 指定任务并行时的cpu数量。如果为-1则使用所有可用的CPU。
#random_state: 一个整数或者一个RandomState实例或者None。
#如果是整数, 则它指定了随机数生成器的种子, 如果为RandomState实例, 则指定了随机数生成器。如果为None则使用默认是随机数生成器。
#solver: 一个字符串, 指定了求解最优化问题的算法, 可以为如下值: newton-cg: 使用牛顿法; lbfgs: 使用L-BFGS拟牛顿法;
#liblinear: 使用liblinear; sag: 使用stochastic average gradient descent算法。注: 对于规模小的数据集, liblinear比较适用, 对于规模大的数据集,
#sag比较适用, 而newton-cg, lbfgs, sag只处理penalty='l2'的情况。tol: 一个浮点数, 指定判断迭代收敛与否的阈值。
#verbose: 一个正数, 用于开启/关闭迭代中间输出日志功能。warm_start: 一个布尔值, 如果为true, 那么使用前一次训练结果继续训练, 否则从头开始训练。
logr.fit(xTrain, yTrain)
```

```
Out[173]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='warn',
n_jobs=None, penalty='l2', random_state=None, solver='warn',
tol=0.0001, verbose=0, warm_start=False)
```

学院君小提示🔔:

上方图片中的标为绿色的文字, 已经对接下来的步骤有了详细的讲解, 在这里对比较重要的知识点进行深入讲解, 其他就不再赘述啦, 大家一定要仔细研读上方文字哦~

(1) C=1.0

C 是逻辑回归公式中正则化的参数，数字越大，正则化越强。正则化将在下面详细介绍。

(2) max_iter=100

有的模型、算法用逻辑回归算法就可以用不同数学办法解决模型，其中有一部分算法就经过一次一次迭代，设置 100 就是指迭代不会超过 100 次。这样模型跑的时间也不会太长，因为到后面迭代次数越来越多，对模型的改进也会越来越少。

(3) penalty='l2'

正则化的一个点，之后讲解。

(4) warm_start=False

如果之前训练过这个模型，设置成 True 的话，会直接用之前的模型进行接下来的训练，略微节省时间。

2. 查看拟合后 logistic regression 模型的 intercept 值

```
In [174]: #查看拟合后logistic regression模型的intercept值  
logr.intercept_
```

```
Out[174]: array([1.53436405])
```

注意！intercept 后一定要加_才会出现结果。

3. 查看拟合后 logistic regression 模型的 coef 值


```
In [175]: #查看拟合后logistic regression模型的coef值  
logr.coef_
```

```
Out[175]: array([[ -0.59398412, -0.01599065, -0.81299548, -0.50226415,  0.00321659,  
                  0.71216015,  0.21910442, -0.06197902, -1.93831832, -0.13443228,  
                  -0.19570416,  0.20499853]])
```

12 个 coef 值对应着 12 个变量的系数。

因为之前说过，逻辑回归模型的公式如下：

Logistic Model

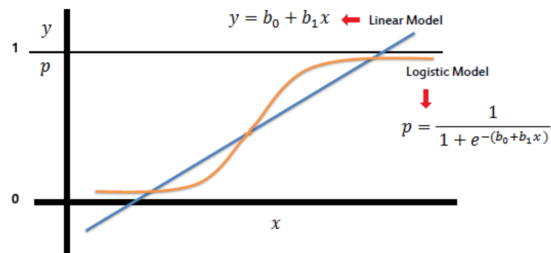

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$

其中 intercept 就是公式中的 b_0 ，12 个不同系数就是 b_1 ， x 就是 12 个变量。

4. 打开名为 “logrformular.png” 的图像

```
In [176]: #打开名为“logrformular.png”的图像
Image(filename='logrformular.png')
```

Out[176]:



```
-(1.6976 - 0.6Pclass - 0.034Age - 0.875SipSb - 0.557Parch + 0.00328*Fare
• 0.715Title + 0.2645FamilySize - 0.018IsAlone - 1.99Sex_Code - 0.123*Embarked_Code
• 0.019AgeBin + 0.22FareBin)
```

在这里把之前 12 个变量填入到公式中。

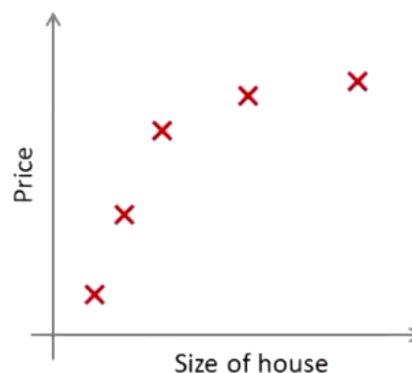
在公式中，1.6976 为最常见的系数，之后每一个可以看到对应着一个变量。其中 SipSb 是兄弟姐妹的数量，如果越大， -0.875SipSb 整个数越大，加了负号， $-(-0.875\text{SipSb})$ 整个数越大，e 的次方越大，分母越大，p 趋向于 0，意味着更趋向于遇难。所以，这就意味着，如果一个乘客兄弟姐妹数量比较多，遇难概率比独自旅游的人会更大。Parch 是指一个人父母孩子数量，和 SipSb 同样，父母孩子数量越多，需要照顾更多的人，自己逃生的几率也会变小。所以这一模型中的一些东西比较符合逻辑思维。还有很典型的比如 Title、Sex_Code，均很符合常理推断。所以逻辑回归的模型可以进行解释，可以将数学公式与正常思维相连接。

5. 正则化

Regularization.

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

$$\min_{\theta} J(\theta)$$



很多情况下模型会过度拟合于训练数据，而在测试集的数据表现的不够好。如何去避免这一问题呢？

此处为大家介绍上图公式：

$h_{\theta}(x^{(i)})$ 就是逻辑回归的公式，结果为 0 或 1，y 就是 yTrain 或 yTest，结果为 0 或 1，如果预测和真实情况相等，相减为 0，如果所有均相等，所有相加起来为 0，我们要让预测值与真实值差异越小越好，如果完全相等，公式算出结果为 0，如果完全相反，则有差异在其中，和精准度很像，但其实是另一种表达方式。

$\lambda \sum_{j=1}^n \theta_j^2$ 即为正则化的一部分。为什么要有正则化呢？下面举例来说明：

如果得到了最后的一个公式，比如为线性回归的公式：

```
In [ ]: 1 y = 2a + 2b
        2 y = a + 3b
        3
        4 if a = b
```

那么 $y_1=y_2$ 。

也就是说如果有两个变量，有某种关联在其中，可以得出两种结果，但均可得到相同答案。但哪一个公式更好呢？在做机器学习的时候，很重要的一点是，公式算出来后要更多的符合普遍数据，而不是说偏向于某个数据。像在 y_2 这里，偏向于 b 这个变量， b 稍有变化，很容易影响 y_2 。相比起来， y_1 就不同，两边更加均衡。

所以，在这两个公式中，如果要加上正则化的话，即为：

```
In [ ]: 1 y1_pred = 2a + 2b
        2 y2_pred = a + 3b
        3 y_true
        4 if a = b
        5 J = (y1_pred - y_true) +
        6
        7 J2 = (y2_pred - y_true) +
        8
```

如果 $a=b$ ， $y_1_pred - y_true$ 和 $y_2_pred - y_true$ 算出结果相等，如何区分差别呢？加入正则化。

正则化中 λ 是系数，为常量，可以先不管。 θ 指 a b 的系数。所以原式可变为：

```
In [ ]: 1 y1_pred = 2a + 2b
        2 y2_pred = a + 3b
        3 y_true
        4 if a = b
        5 J = (y1_pred - y_true) + 4 + 4
        6
        7 J2 = (y2_pred - y_true) + 1 + 9|
        8
```

可以看出差别：J2 比 J 要多， y_2_pred 比起 y_1_pred 不够普遍， y_1_pred 可以更好的 fit 比较正常、比较普遍的数据。区别的方法就是使用正则化。

加入正则化正是能够让模型更少的过度拟合于原有的训练数据，让模型在测试集中有更好的表现。这就是避免过度拟合的一种方法——Regularization。

二、决策树

1. 因为决策树长得和树相似，所以在 Python 中可以使用 graphviz 对其进行画图

```
In [177]: #载入graphviz数据包, 以及从sklearn中载入tree数据包: graphviz是AT&T实验室开源的画图工具,
import graphviz
from sklearn import tree
```

2. 创建决策树对象

```
In [229]: #创建名为dtree的决策树对象
dtree = DecisionTreeClassifier()
```

3. fit 训练数据

```
In [235]: #调用DecisionTree的fit方法对数据集进行拟合
#class_weight:dict,list of dicts,"Balanced" or None,可选 (默认为None)
#criterion:string类型, 可选 (默认为"gini") 衡量分类的质量。支持的标准有: "gini"代表的是Gini impurity(不纯度);
#"entropy"代表的是information gain (信息增益)。
#max_depth:int or None,可选 (默认为"None") 表示树的最大深度。如果是"None",则节点会一直扩展直到所有的叶子都是纯的或者所有的叶子节点都包含
#少于min_samples_split个样本点。忽视max_leaf_nodes是不是为None。
#max_features:int,float,string or None 可选 (默认为None), 在进行分类时需要考虑的特征数。None, max_features=n_features 注意: 至少找到一个
#样本点有效的被分类时, 搜索分类才会停止。
#max_leaf_nodes:int,None 可选 (默认为None) 在最优方法中使用max_leaf_nodes构建一个树。最好的节点是在杂质相对减少。
#如果是None则对叶节点的数目没有限制。如果不是None则不考虑max_depth。
#min_impurity_decrease: float, optional (default=0.)如果该分裂导致杂质的减少大于或等于该值, 则将分裂节点。
#min_samples_leaf:int,float,可选 (默认为1) 一个叶节点所需要的最小样本数。
#min_samples_split:int,float,可选 (默认为2) 区分一个内部节点需要的最少的样本数。
#min_weight_fraction_leaf:float,可选 (默认为0) 一个叶节点的输入样本所需要的最小的加权分数。
#persort:bool,可选 (默认为False) 是否预分类数据以加速训练时最好分类的查找。在有大数据集的决策树中, 如果设为true可能会减慢训练的过程。
#当使用一个小数据集或者一个深度受限的决策树中, 可以加速训练的过程。
#random_state:int,RandomState instance or None; 如果是None, 随机数字发生器是np.random使用的RandomState instance。
#splitter:string类型, 可选 (默认为"best") 一种用来在节点中选择分类的策略。支持的策略有"best", 选择最好的分类, "random"选择最好的随机分类。
dtree.fit(XTrain,yTrain)
```

```
Out[235]: 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')
```

4. 利用 tree 的 export_graphviz 以 DOT 形式提取决策树

```
In [197]: #利用tree的export_graphviz以DOT形式提取决策树
dot_data = tree.export_graphviz(dtree, out_file=None, feature_names=XTrain.columns, filled=True, rounded=True)
```

5. 将 DOT 形式的决策树源码字符串形式

```
In [198]: #将DOT形式的决策树源码字符串形式
graph = graphviz.Source(dot_data)
```

6. 用 pdf 存储源码

```
In [199]: #用pdf存储源码
graph.render('dtree')
```

```
Out[199]: 'dtree_res.pdf'
```

形成了非常巨大的树，涵盖了所有的 feature。

决策树比较所有变量，选取最好的能够区分死亡和存活的 feature 进行运用，因此

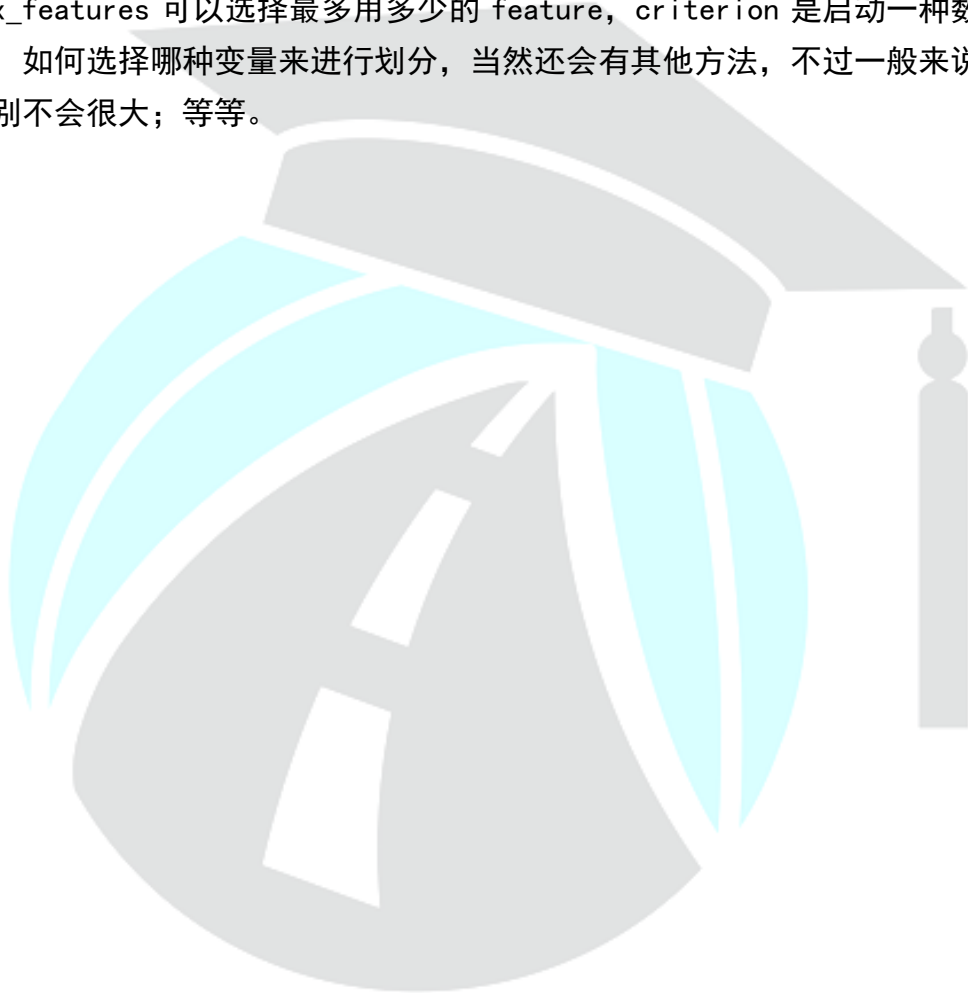
在树中越上方的这些方框相比于下方更加重要。

树过于巨大，但只要加上 `max_depth`，比如=5，设置的是树的层数。树只有 5 层，可以比较清晰的看到划分。

这就是决策树的绘图方法，尝试思考，为什么会这么划分？

7. 重要参数

决策树中也有一些重要参数可以进行调整，比如 `max_depth` 调整模型最深可以多少层，`max_features` 可以选择最多用多少的 feature，`criterion` 是启动一种数学计算方法，如何选择哪种变量来进行划分，当然还会有其他方法，不过一般来说大同小异区别不会很大；等等。



三、随机森林

在很多时候随机森林整体表现会是更好的一个模型。

1. 创建名为 rf_10 的随机森林对象

```
In [200]: #创建名为rf_10的随机森林对象，森林里决策树的数目为10  
rf_10 = RandomForestClassifier(n_estimators=10)
```

`n_estimators=10`，意义为这个随机森林中随机生成了 10 个不同的决策树，让其进行整合得到结果。

2. 训练模型

```
In [201]: #调用RandomForestClassifier的fit方法对数据集进行拟合  
#bootstrap=True: 是否有放回的采样。  
#max_features: 选择最适属性时划分的特征不能超过此值。当为整数时，即最大特征数；当为小数时，训练集特征数*小数；  
#if "auto", then max_features=sqrt(n_features).  
#n_estimators=10: 决策树的个数，越多越好，但是性能就会越差，至少100左右  
#n_jobs=1: 并行job个数。这个在ensemble算法中非常重要，尤其是bagging（而非boosting，因为boosting的每次迭代之间有影响，所以很难进行并行化），  
#因为可以并行从而提高性能。1=不并行；n: n个并行；-1: CPU有多少core，就启动多少job。  
#oob_score=False: oob (out of band, 带外) 数据，即：在某次决策树训练中没有被bootstrap选中的数据。  
#verbose:(default=0) 是否显示任务进程  
#warm_start=False: 热启动，决定是否使用上次调用该类的结果然后增加新的。  
rf_10.fit(xTrain,yTrain)
```

```
Out[201]: 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=10, n_jobs=None,  
oob_score=False, random_state=None, verbose=0,  
warm_start=False)
```

3. 查看刚才创建的决策树中的第五棵树

```
In [203]: #查看刚才创建的决策树中的第五棵树  
rf_5 = rf_10.estimators_[5]
```

随机森林是不同树组建在一起，和决策树有所差别，不可以直接画图，而是可以画其中一棵树。

4. 同上决策树画图方法对其进行画图

```
In [204]: #利用tree的export_graphviz以DOT形式提取决策树  
dot_data = tree.export_graphviz(rf_5, out_file=None, feature_names=xTrain.columns, filled=True, rounded=True)
```

```
In [205]: #将DOT形式的决策树源码字符串形式  
graph = graphviz.Source(dot_data)
```

```
In [207]: #用pdf存储源码  
graph.render('rftree')
```

```
Out[207]: 'rftree.pdf'
```

也是非常巨大的一棵树。

随机森林中树的选择是随机的，第 5 颗树可能是从 12 个变量中随机选择一部分进行模型的构建。所以用不同数据进行计算，得到的结果不同。

5. 重要参数

随机森林中的 feature 和决策树很像,几乎一模一样,但就是多了 `n_estimators` 等,因为是多个决策树组成,所以必须加上 `n_estimators`。随机森林的 `max_features='auto'`。什么是 auto 呢? 为 \sqrt{n} 。会选取这样一些数据进行操作。



四、Overfitting 过度拟合

无论在机器学习还是深度学习建模当中都可能会遇到两种最常见结果，一种叫过拟合（over-fitting）另外一种叫欠拟合（under-fitting）。

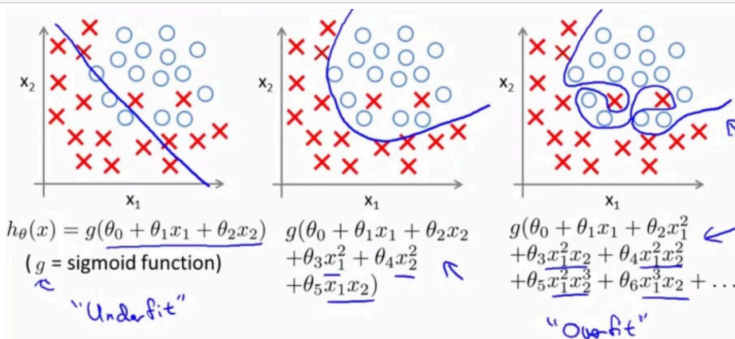
所谓过拟合（over-fitting）其实就是所建的机器学习模型或者是深度学习模型在训练样本中表现得过于优越，导致在验证数据集以及测试数据集中表现不佳。

过拟合就是学到了很多没必要的特征。

首先创建模型区分○和×，诠释什么是过度拟合。

```
In [208]: #导入图像，诠释什么是过度拟合  
Image(filename='overfit.png')
```

Out[208]:



第三个模型即为过度拟合。运用这一模型会出现很多错误的 label，出现错误标签，造成错误预测。训练集的数据正确率相当高，如果加了测试集的数据，正确率会降低。

了解了过度拟合的概念，再回到决策树模型中来看这一问题。

```
In [241]: #利用decisiontree的predict预测测试集和训练集  
y_pred_train = dtree.predict(xTrain)  
y_pred_test = dtree.predict(xTest)
```

```
In [242]: #利用numpy数据包的mean取平均，评估预测的精确度  
np.mean(y_pred_train == yTrain), np.mean(y_pred_test == yTest)
```

Out[242]: (0.9887640449438202, 0.7653631284916201)

出现问题的原因在于树分的太细了，但其实很多很细的分类只在训练集数据中才会体现其作用，在测试集中很容易造成误导，怎么在这种情况下避免这种问题呢？最简单的方法就是把树的最大深度变小。

```
In [246]: #创建名为dtree2的决策树对象，规定树的最大深度为5，构成一个内部节点的样本最少为5个  
dtree2 = DecisionTreeClassifier(max_depth=5, min_samples_split=5)
```

```
In [248]: #利用DecisionTreeClassifier的fit方法拟合xy训练集  
dtree2.fit(xTrain, yTrain)
```

Out[248]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=5, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

```
In [249]: #利用numpy数据包的mean取平均，评估预测的精确度  
np.mean(dtree2.predict(xTrain) == yTrain), np.mean(dtree2.predict(xTest) == yTest)
```

Out[249]: (0.8525280898876404, 0.8379888268156425)

把树的最大深度变小，使得模型更加具有普适性，而不是只针对训练集，所以在测试集表现更好。这也是非常重要的知识点。，同时是解决过度拟合问题的第二种方法。

学院君小提示🕒：

有很多参数都可以一个一个去试，试了之后才知道对于数据哪些参数是最好的。所以有“调参”这一说法。大家可以大胆尝试哦～

在这节课我们训练了这些模型，下节课会看最终得到的结果，并为大家讲解在机器学习中一些方法和比较常见的可以使用的网站等其他知识点，下节课见\(^o^)/



四、作业

在理解了随机森林这个模型后，请同学们自己去调试一下几个重要的参数，`max_depth`, `n_estimators`, `min_samples_split`, 看看这几个参数如何组合会得到更好的结果？



五、第四讲作业答案及解析

答案：

一般来讲，k 设置越大，整个对数据的比较就会越精确，k 最大可以设置为 $n-1$ ，这样的话，即为用所有的数据 ($n-1$) 去 train，预测余下一行数据。k 变大的话会增大计算量，现实生活中，在很多情况下，数据会很大，train 一个模型要花很多时间，所以计算量有时候也需要考虑进去。当然，具体最佳的 k 值在不同的 project 中会不一样。

解析：

一般来说 k 的值设置越大，数据的比较就会更多，数据平均的值就会更加稳定，相对来说就更加可靠，所以可以说 k 越大越好。

k 最大可以设置为： $k = n - 1$, n is $\text{len}(x\text{Train})$

n 就是数据的行数，举例来说，比如数据有 100 行，k 最大设置为 99，在这种情况下每次训练 99 行数据，预测最后 1 行数据，每次循环下来，模型一共运行 100 次，再取 100 次的平均结果。在这种情况下 k 是最大值。

但在现实情况下，不会追求这种极端情况。因为 k 越大，计算量会增加很多，现实生活中的数据大多数情况下都会很大，k 越大会很消耗计算量，在很多时候得到模型的速度也是非常重要的，并不是说可以等很久得到答案，有时候需要快速得到结果再进行下一步的操作。



扫码关注棕榈学院，解锁更多精彩课程