**个人信用风险评估**

# 1简介

2007-2008年的全球金融危机凸显了透明度和严密性在银行业务中的重要性。由于信贷供应收到了限制，所以银行正日益紧缩其贷款体系，转向机器学习来更准确地识别高风险贷款。

决策树模型准确性高且可解释性好，所以被广泛地应用在银行业。在很多国家，政府机构会密切监控贷款业务，所以银行需要明确地解释为什么一个申请者的贷款申请被拒绝或者批准。这种可解释性对于贷款申请者也是很重要的，申请者需要知道为什么自己的信用级别不符合银行的要求。

通过构建自动化的信用评分模型，以在线方式进行即时的信贷审批能够为银行节约很多人工成本。本案例，将使用C5.0决策树算法建立一个简单的个人信用风险评估模型。

# 2准备

## 2.1实验环境

本实验延续以往，使用Anaconda + Jupyter Notebook环境。

导入类库。

*# coding=utf-8*import pandas as pd  
*# 用来显示图片的工具包*from IPython.display import Image  
*# 实现在内存缓冲区中读写数据*from six import StringIO  
*# 引入模型评价函数*from sklearn import model\_selection, tree, metrics  
*# 引入决策树模型*from sklearn.tree import DecisionTreeClassifier  
*# 生成图表*import pydotplus

## 2.2数据准备

使用UCI上的德国信用数据集。该数据集包含了1000个贷款信息，每一个贷款有20个自变量和一个类变量记录该笔贷款是否违约。我们将使用该数据集构建模型来预测贷款是否违约。

将实验数据集“credit.csv”拷贝到此前在D盘建立的input文件夹。

对数据集各个属性的说明如下：

|  |  |
| --- | --- |
| 属性 | 说明 |
| checking\_balance | 支票余额 |
| months\_loan\_duration | 贷款期限（月） |
| credit\_history | 信用记录 |
| purpose | 贷款目的 |
| amount | 贷款申请额度 |
| savings\_balance | 储蓄余额 |
| employment\_length | 雇佣时长 |
| installment\_rate | 分期付款利率 |
| personal\_status | 个人状况 |
| other\_debtors | 其他债务人 |
| residence\_history | 居住历史 |
| property | 财产 |
| age | 年龄 |
| installment\_plan | 分期付款计划 |
| housing | 住房情况 |
| existing\_credits | 现有信用 |
| job | 工作 |
| dependents | 家属 |
| telephone | 是否留下电话 |
| foreign\_worker | 是否是外国工人 |
| default | 是否违约 |

## 2.3数据读入

使用Pandas的read\_csv()函数读取并查看数据。

**实验内容**：

*# 读取数据集 read\_cscv  
# 输出数据集前十行 head(10)*

*# 查看数据的规模 shap*

# 3数据预处理

## 3.1查看数据特点

**实验内容**：

*# 查看支票余额属性各个取值的分布*

*# 查看储蓄账户余额的分布*

*# 查看贷款期限和贷款申请额度的分布情况*

*# 获取贷款期限和贷款申请额度的中位数*

*# 查看数据集中违约和非违约样本的数量*。

## 3.2数据编码

Scikit-learn中的决策树模型要求输入的自变量为数值型，在划分数据之前，我们还需要将数据中字符串形式的变量使用整数进行编码。

**实验内容：**

*# 创建编码所用的数据字典*  
*# 获取编码的属性集  
# 定义编码规则  
# 执行编码  
# 查看编码，输出前五行数据*

## 3.3划分数据集

在正式建模之前，我们需要将数据集分为训练集和测试集两部分。其中训练集用来构建决策树模型，测试集用来评估模型性能。 我们将使用70%数据作为训练数据，30%作为训练数据。

**实验内容：**

*# 确定因变量*y = credit[**'default'**]  
*# 确定自变量*X = credit.loc[:, **'checking\_balance'**:**'foreign\_worker'**]  
*# 划分训练集和测试集，random\_state=1表示先打乱顺序再划分，测试集占比30%*X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.3, random\_state=1)

验证训练集和测试集中，违约贷款的比例是否接近。

*# 查看训练集中违约和非违约样本的分布*y\_train.value\_counts() / len(y\_train)

*# 查看测试集中违约和非违约样本的分布*y\_test.value\_counts() / len(y\_test)

# 4模型构建和训练

我们将使用Scikit-learn中的DecisionTreeClassifier算法来训练决策树模型。 DecisionTreeClassifier算法位于sklearn.tree包，首先将其导入，然后调用fit()方法进行模型训练。

关于scikit-learn的决策树分类模型，其可以设置很多参数，如下：

DecisionTreeClassifier(criterion=**"gini"**,  
 splitter=**"best"**,  
 max\_depth=None,  
 min\_samples\_split=2,  
 min\_samples\_leaf=1,  
 min\_weight\_fraction\_leaf=0.,  
 max\_features=None,  
 random\_state=None,  
 max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.,  
 min\_impurity\_split=None,  
 class\_weight=None,  
 presort=False)

1.criterion:string, optional (default="gini")

(1).criterion='gini',分裂节点时评价准则是Gini指数。

(2).criterion='entropy',分裂节点时的评价指标是信息增益。

2.max\_depth:int or None, optional (default=None)。指定树的最大深度。

如果为None，表示树的深度不限。直到所有的叶子节点都是纯净的，即叶子节点中所有的样本点都属于同一个类别。或者每个叶子节点包含的样本数小于min\_samples\_split。

3.splitter:string, optional (default="best")。指定分裂节点时的策略。

(1).splitter='best',表示选择最优的分裂策略。

(2).splitter='random',表示选择最好的随机切分策略。

4.min\_samples\_split:int, float, optional (default=2)。表示分裂一个内部节点需要的做少样本数。

(1).如果为整数，则min\_samples\_split就是最少样本数。

(2).如果为浮点数(0到1之间)，则每次分裂最少样本数ceil(min\_samples\_split \* n\_samples)

5.min\_samples\_leaf: int, float, optional (default=1)。指定每个叶子节点需要的最少样本数。

(1).如果为整数，则min\_samples\_split就是最少样本数。

(2).如果为浮点数(0到1之间)，则每个叶子节点最少样本数为ceil(min\_samples\_leaf \* n\_samples)

6.min\_weight\_fraction\_leaf:float, optional (default=0.)

指定叶子节点中样本的最小权重。

7.max\_features:int, float, string or None, optional (default=None)搜寻最佳划分的时候考虑的特征数量。

(1).如果为整数，每次分裂只考虑max\_features个特征。

(2).如果为浮点数(0到1之间)，每次切分只考虑int(max\_features \* n\_features)个特征。

(3).如果为'auto'或者'sqrt',则每次切分只考虑sqrt(n\_features)个特征

(4).如果为'log2',则每次切分只考虑log2(n\_features)个特征。

(5).如果为None,则每次切分考虑n\_features个特征。

(6).如果已经考虑了max\_features个特征，但还是没有找到一个有效的切分，那么还会继续寻找下一个特征，直到找到一个有效的切分为止。

8.random\_state:int, RandomState instance or None, optional (default=None)

(1).如果为整数，则它指定了随机数生成器的种子。

(2).如果为RandomState实例，则指定了随机数生成器。

(3).如果为None，则使用默认的随机数生成器。

9.max\_leaf\_nodes: int or None, optional (default=None)。指定了叶子节点的最大数量。

(1).如果为None,叶子节点数量不限。

(2).如果为整数，则max\_depth被忽略。

10.min\_impurity\_decrease:float, optional (default=0.)如果节点的分裂导致不纯度的减少(分裂后样本比分裂前更加纯净)大于或等于min\_impurity\_decrease，则分裂该节点。

加权不纯度的减少量计算公式为：

min\_impurity\_decrease=N\_t / N \* (impurity - N\_t\_R / N\_t \* right\_impurity- N\_t\_L / N\_t \* left\_impurity)

其中N是样本的总数，N\_t是当前节点的样本数，N\_t\_L是分裂后左子节点的样本数，N\_t\_R是分裂后右子节点的样本数。impurity指当前节点的基尼指数，right\_impurity指分裂后右子节点的基尼指数。left\_impurity指分裂后左子节点的基尼指数。

11.min\_impurity\_split:float

树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值，节点将分裂，否则它是叶子节点。

这个参数已经被弃用。用min\_impurity\_decrease代替了min\_impurity\_split。

12.class\_weight:dict, list of dicts, "balanced" or None, default=None

类别权重的形式为{class\_label: weight}

(1).如果没有给出每个类别的权重，则每个类别的权重都为1。

(2).如果class\_weight='balanced'，则分类的权重与样本中每个类别出现的频率成反比。计算公式为：n\_samples / (n\_classes \* np.bincount(y))

(3).如果sample\_weight提供了样本权重(由fit方法提供)，则这些权重都会乘以sample\_weight。

13.presort:bool, optional (default=False)

指定是否需要提前排序数据从而加速训练中寻找最优切分的过程。设置为True时，对于大数据集会减慢总体的训练过程；但是对于一个小数据集或者设定了最大深度的情况下，会加速训练过程。

*# 创建模型  
# min\_samples\_leaf可以为整数或者浮点数，默认为1。它指定了每个叶子节点包含的最少样本数。  
# 如果为浮点数(0到1之间)，每个叶子节点包含的最少样本数为ceil(min\_samples\_leaf \* n\_samples)*credit\_model = DecisionTreeClassifier(min\_samples\_leaf=6, random\_state=1)  
*# 训练模型*credit\_model.fit(X\_train, y\_train)

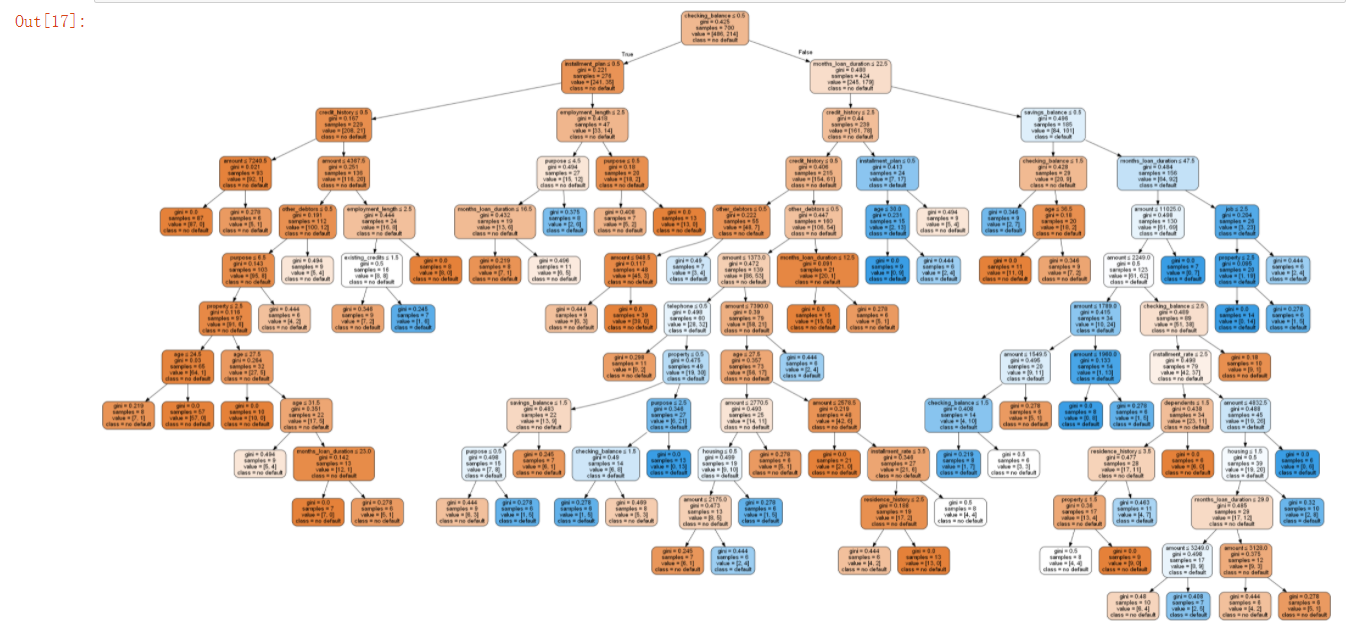
现在，credit\_model就是我们训练得到的决策树模型。可以通过可视化将其展示出来。

绘制决策树模型的图形时，需要尤其注意pydotplus的安装和配置，此处极容易出问题。Pydotplus是pydot的升级版本，pydot已停止研发，会出现版本不适配的问题，故而推荐大家都是用pydotplus进行实验。

pydot和pydotplus都是基于Graphviz实现图形绘制的，故而需要单独安装Graphviz，在官网下载和安装就好，注意安装完成后，需要配置环境变量，将Graphviz目录下的bin文件夹路径添加到系统环境变量path中。

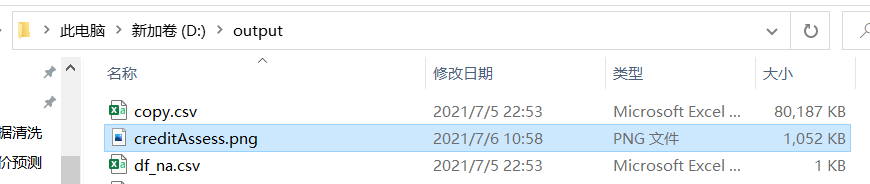
需注意，就算配置好环境变量，pydotplus调用Graphviz时也可能会报错，需要在代码中绘制前再添加一遍，才能确保不出问题。

*# 获取决策树的数据*dot\_data = StringIO()  
*# 决策树构建*tree.export\_graphviz(credit\_model, out\_file=dot\_data,  
 feature\_names=X\_train.columns,  
 class\_names=[**'no default'**, **'default'**],  
 filled=True, rounded=True,  
 special\_characters=True)  
graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())  
  
*# 此处用于将Graphviz目录添加到环境变量中  
# 注意，配置好环境变量运行报错，可以加上此语句再运行，查看结果*import os  
os.environ[**"PATH"**] += os.pathsep + **'D:/Program Files/Graphviz/bin/'***# 绘制决策树*Image(graph.create\_png())



因为图片较大，大家也可以将图片保存到本地查看。

*# 保存图片到本地*graph.write\_png(**'D:/output/creditAssess.png'**)



# 5模型性能评估

在实际应用中，模型的预测正确率不高，很难将其应用到实时的信贷评审过程。为了客观评价模型的好坏，我们需要对模型进行性能评估。

为了将我们训练好的决策树模型应用于测试数据，我们使用predict()函数，代码如下：

*# 推测测试集标签*credit\_pred = credit\_model.predict(X\_test)

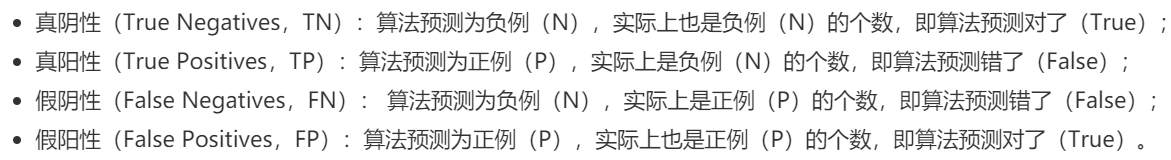
现在，我们得到了决策树模型在测试数据上的预测结果，通过将预测结果和真实结果进行对比可以评估模型性能。 可以使用sklearn.metrics包中的classification\_report()和confusion\_matrix()函数，展示模型分类结果。

准确率（正确率）即为预测正确的样本占全部样本的比例，介于0和1之间。如果该值接近0，则认为是性能不佳；而如果该值接近1，则认为其性能良好。 它是最简单易懂的指标之一。

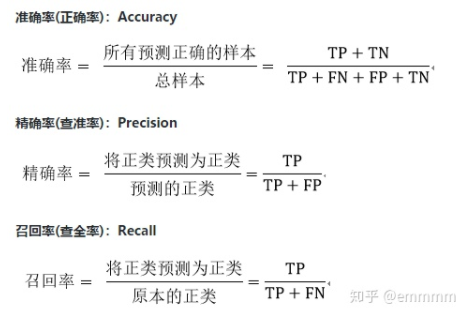
召回率（查全率）=真正例/（真正例 + 假反例），即本案例中我们正确判定的违约样本占全部违约样本的比例。

精确率（查准率）=真正例/（真正例+假正例），即本案例中我们判定违约的样本中，真正违约样本的比例。

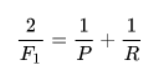
要解释清楚这三个值的差别，得先解释混淆矩阵，二分类问题的混淆矩阵由 4 个数构成。



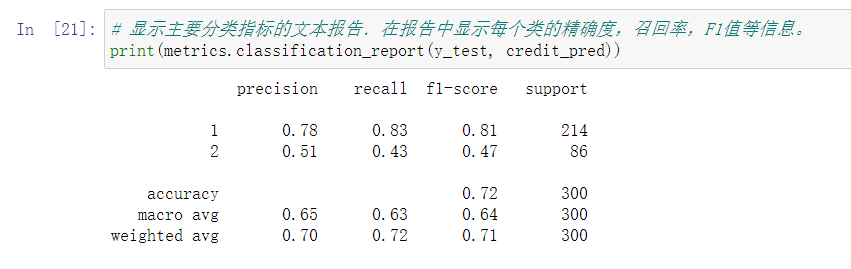




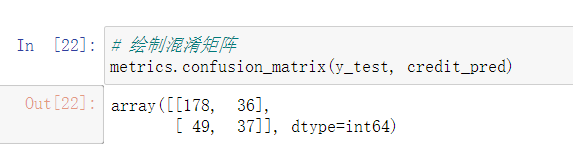
精准率和召回率是此消彼长的，即精准率高了，召回率就下降，在一些场景下要兼顾精准率和召回率，就有 F1 score。F1值是来综合评估精确率和召回率，当精确率和召回率都高时，F1也会高。



*# 显示主要分类指标的文本报告．在报告中显示每个类的精确度，召回率，F1值等信息。*print(metrics.classification\_report(y\_test, credit\_pred))



*# 绘制混淆矩阵*metrics.confusion\_matrix(y\_test, credit\_pred)



*# 获取分类准确率分数，即所有分类正确的百分比。*print(metrics.accuracy\_score(y\_test, credit\_pred))



在300个贷款申请测试数据中，模型的预测正确率（Accuracy）为71.7%。 214个未违约贷款中，模型正确预测了82%。86个违约贷款中，模型正确预测出了43%。下面，我们看看是否能够进一步改善模型的性能。

# 6模型性能提升

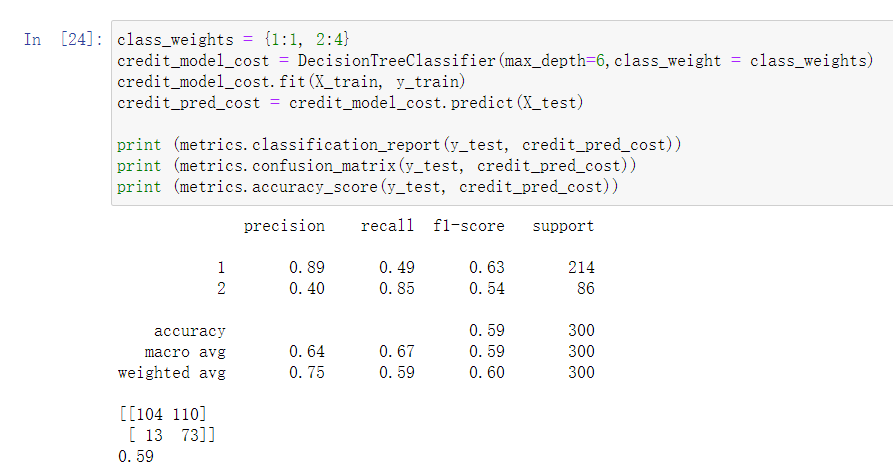
在本案例中，如果一个模型将所有的贷款都预测为“未违约”，此时模型的正确率将为72%，而该模型是一个完全无用的模型。 上节中我们建立的模型，正确率为70.7%，但是对于违约贷款的识别性能很差。 我们可以通过创建一个代价矩阵定义模型犯不同错误时的代价。

假设我们认为一个贷款违约者给银行带来的损失是银行错过一个不违约的贷款带来损失的4倍，则未违约和违约的代价权重可以定义为：

*# 认为一个贷款违约者给银行带来的损失是银行错过一个不违约的贷款带来损失的4倍*class\_weights = {1: 1, 2: 4}  
credit\_model\_cost = DecisionTreeClassifier(max\_depth=6, class\_weight=class\_weights)  
credit\_model\_cost.fit(X\_train, y\_train)

下面我们来测试一下这个模型的性能。

credit\_pred\_cost = credit\_model\_cost.predict(X\_test)  
print(metrics.classification\_report(y\_test, credit\_pred\_cost))  
print(metrics.confusion\_matrix(y\_test, credit\_pred\_cost))  
print(metrics.accuracy\_score(y\_test, credit\_pred\_cost))



可见，模型的整体正确率下降为59%，但是此时的模型能将86个违约贷款中的73个正确识别，识别率为85%。

因为对违约贷款的识别能力增强，模型的整体价值提高了。

同学们可以尝试修改模型的各个参数，建立性能更好的模型。