**决策树和 K 近邻分类**

1. ****实验目标****

学会运用决策树和最近邻方法在分类任务上构建模型，并通过交叉验证对模型进行调优。

1. ****实验内容****

* 决策树
* 最近邻方法
* 交叉验证

1. ****实验步骤及结果截图****

二元分类问题的熵和基尼不确定性为：

S = -p\_+ \log\_2{p\_+} -p\_- \log\_2{p\_-} = -p\_+ \log\_2{p\_+} -(1 - p\_{+}) \log\_2{(1 - p\_{+})}*S*=−*p*+​log2​*p*+​−*p*−​log2​*p*−​=−*p*+​log2​*p*+​−(1−*p*+​)log2​(1−*p*+​)

G = 1 - p\_+^2 - p\_-^2 = 1 - p\_+^2 - (1 - p\_+)^2 = 2p\_+(1-p\_+)*G*=1−*p*+2​−*p*−2​=1−*p*+2​−(1−*p*+​)2=2*p*+​(1−*p*+​)

其中 p\_+*p*+​ 是对象具有标签 + 的概率。

* 以 p\_+p+​ 为坐标，绘制上面两个函数的图像

import warnings from matplotlib import pyplot as plt import numpy as np import pandas as pd import seaborn as sns sns.set() warnings.filterwarnings('ignore')

plt.figure(figsize=(6, 4))

xx = np.linspace(0, 1, 50)

plt.plot(xx, [2 \* x \* (1-x) for x in xx], label='gini')

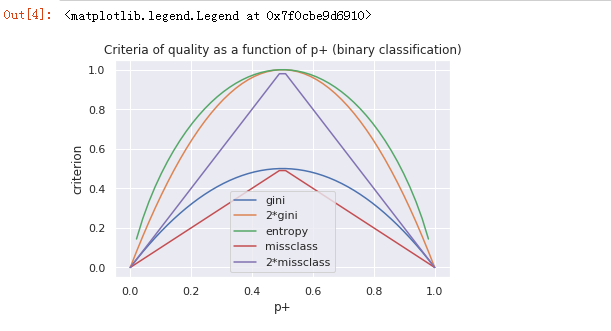
plt.plot(xx, [4 \* x \* (1-x) for x in xx], label='2\*gini')

plt.plot(xx, [-x \* np.log2(x) - (1-x) \* np.log2(1 - x) for x in xx], label='entropy')

plt.plot(xx, [1 - max(x, 1-x) for x in xx], label='missclass')

plt.plot(xx, [2 - 2 \* max(x, 1-x) for x in xx], label='2\*missclass') plt.xlabel('p+') plt.ylabel('criterion')

plt.title('Criteria of quality as a function of p+ (binary classification)') plt.legend()



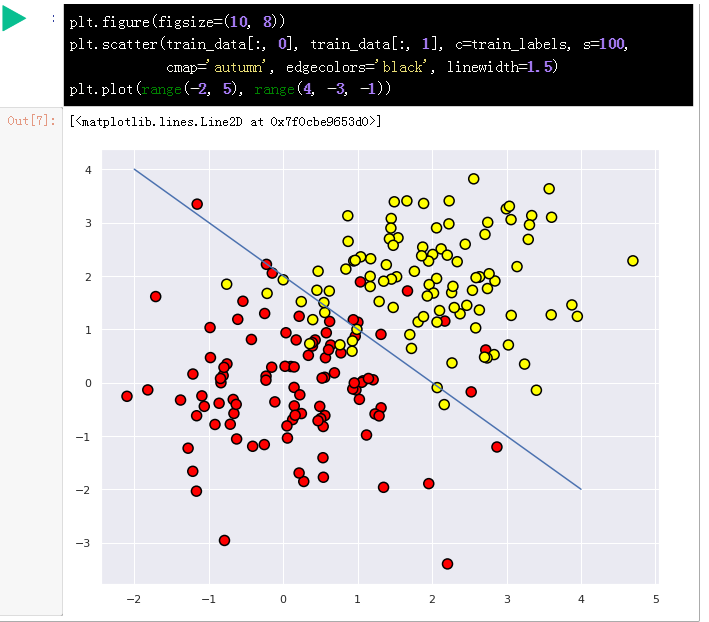
上图可见，熵的图像和两倍的基尼不确定性图像非常接近。因此，在实践中，这两个指标的效果基本上是一样的。

* 下面用一棵决策树拟合一些合成数据。这些合成数据属于两个不同的类别，这两个类别的均值不同，但都呈现正态分布。

# 第一类 np.random.seed(17) train\_data = np.random.normal(size=(100, 2)) train\_labels = np.zeros(100)

# 第二类 train\_data = np.r\_[train\_data, np.random.normal(size=(100, 2), loc=2)] train\_labels = np.r\_[train\_labels, np.ones(100)]

plt.figure(figsize=(10, 8)) plt.scatter(train\_data[:, 0], train\_data[:, 1], c=train\_labels, s=100, cmap='autumn', edgecolors='black', linewidth=1.5) plt.plot(range(-2, 5), range(4, -3, -1))

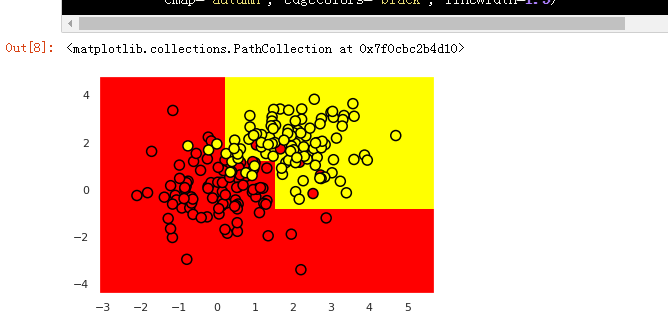


* 下面训练一棵 sklearn 决策树，区分这两类数据点。最后可视化所得的边界。

from sklearn.tree import DecisionTreeClassifier # 编写一个辅助函数，返回之后的可视化网格

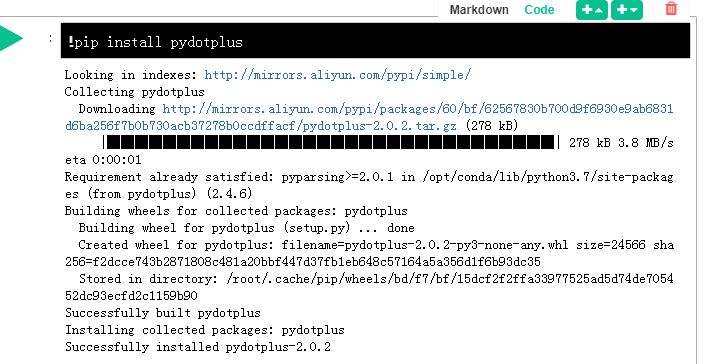
def get\_grid(data): x\_min, x\_max = data[:, 0].min() - 1, data[:, 0].max() + 1 y\_min, y\_max = data[:, 1].min() - 1, data[:, 1].max() + 1 return np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

# max\_depth参数限制决策树的深度 clf\_tree = DecisionTreeClassifier(criterion='entropy', max\_depth=3, random\_state=17) # 训练决策树 clf\_tree.fit(train\_data, train\_labels) # 可视化 xx, yy = get\_grid(train\_data) predicted = clf\_tree.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape) plt.pcolormesh(xx, yy, predicted, cmap='autumn') plt.scatter(train\_data[:, 0], train\_data[:, 1], c=train\_labels, s=100, cmap='autumn', edgecolors='black', linewidth=1.5)



* 通过 pydotplus 和 export\_graphviz 库我们可以方便的看到决策树本身是怎样的。使用 StringIO()函数开辟一个缓存空间保存决策树，通过 export\_graphviz() 函数以 DOT 格式导出决策树的 GraphViz 表示，然后将其写入 out\_file 中。使用 graph\_from\_dot\_data() 函数读入数据并通过 Image() 函数显示决策树。

!pip install pydotplus # 安装必要模块



from ipywidgets

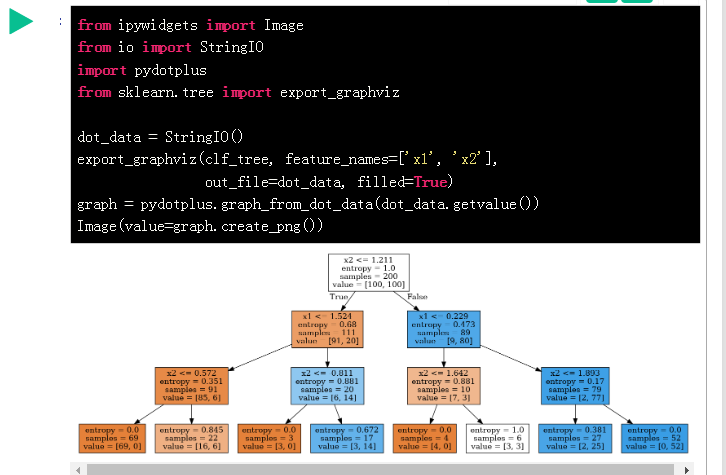
import Image

from io import StringIO

import pydotplus

from sklearn.tree

import export\_graphviz dot\_data = StringIO() export\_graphviz(clf\_tree, feature\_names=['x1', 'x2'], out\_file=dot\_data, filled=True) graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue()) Image(value=graph.create\_png())

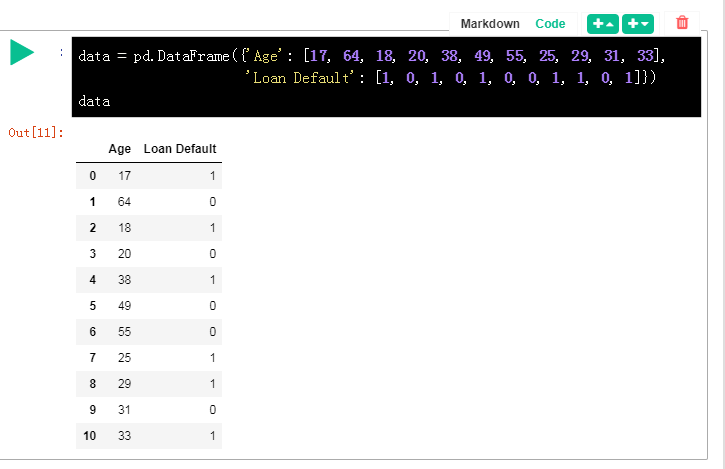


### **决策树如何应用到数值特征？**

* 假设有一个数值特征「年龄」，该特征有大量的唯一值。决策树将通过查看「年龄 < 17」、「年龄 < 22.87」这样的二元属性寻找最好的分割，分割的好坏由某种信息增益标准衡量。但在构建树的每一步中，会有过多的二元属性可供选择，比如「薪水」同样能以很多方式进行分割，为了解决这一问题，我们经常使用启发式算法来限制选择的属性数量。

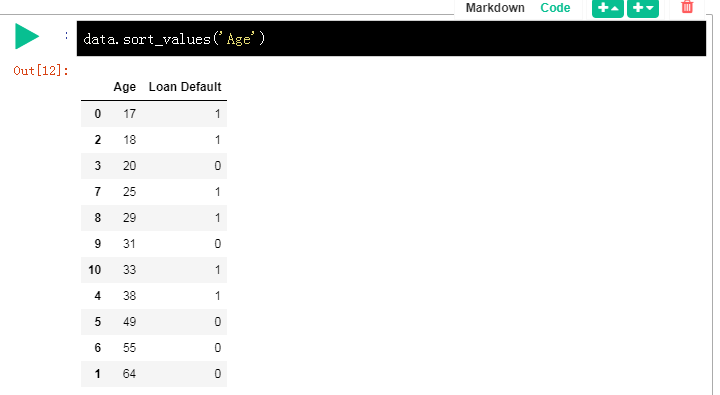
看一个例子，假设有如下数据集：

data = pd.DataFrame({'Age': [17, 64, 18, 20, 38, 49, 55, 25, 29, 31, 33], 'Loan Default': [1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1]}) data



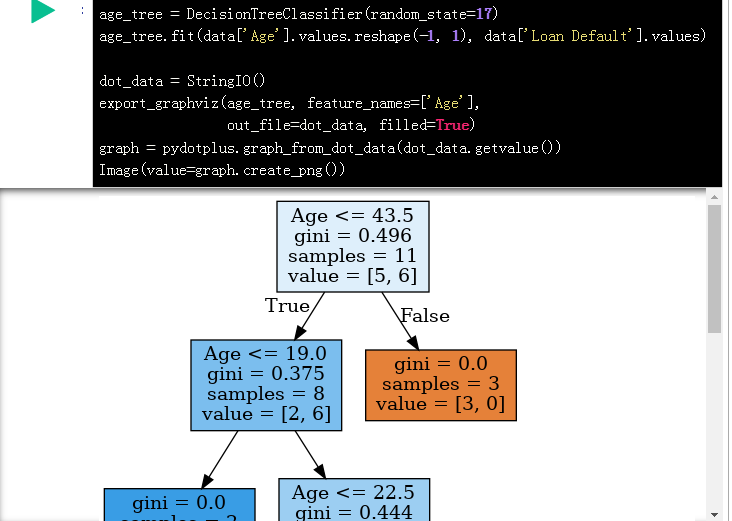
使用 sort\_values() 方法根据年龄进行升序排列。

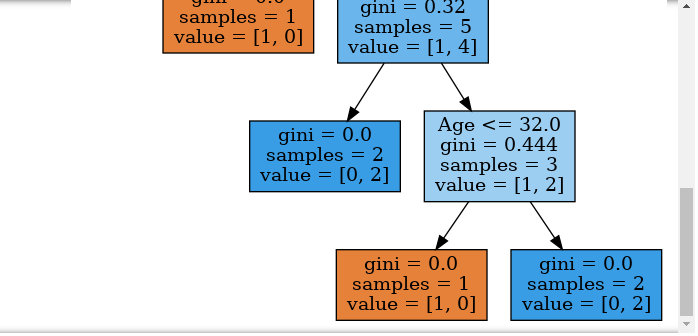
data.sort\_values('Age')



训练一个决策树模型，并可视化。

age\_tree = DecisionTreeClassifier(random\_state=17) age\_tree.fit(data['Age'].values.reshape(-1, 1), data['Loan Default'].values) dot\_data = StringIO() export\_graphviz(age\_tree, feature\_names=['Age'], out\_file=dot\_data, filled=True) graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue()) Image(value=graph.create\_png())

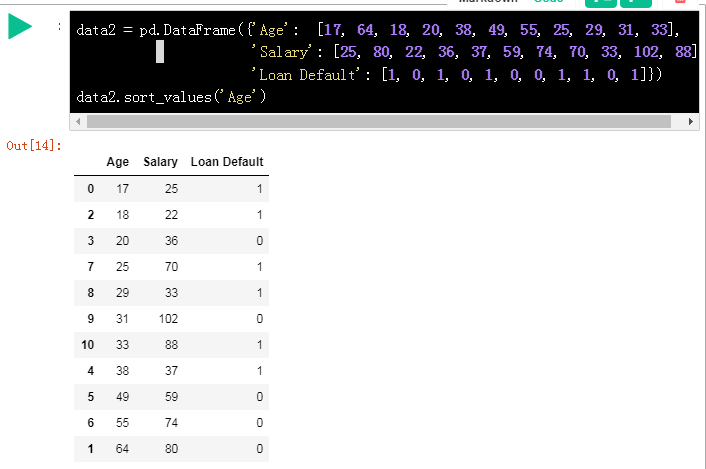




上图可见，该决策树使用以下 5 个值来评估年龄：43.5、19、22.5、30、32。如果你仔细观察，你会发现它们就是目标变量（Loan Default）出现变化（从 1「切换」到 0 或从 0「切换」到 1）时那两个年龄的平均值。比如，一个 38 岁的客户没能偿还贷款（目标变量为 1），而一个 49 岁的客户还贷了（目标变量为 0），那么树使用的评估值就是 38 和 49 的均值，即 43.5。树寻找那些目标变量发生变化的值，以此作为「切割」的阈值。

下面考虑一个更复杂的例子，把「薪水」变量（以千美元每年为单位）加入数据集。

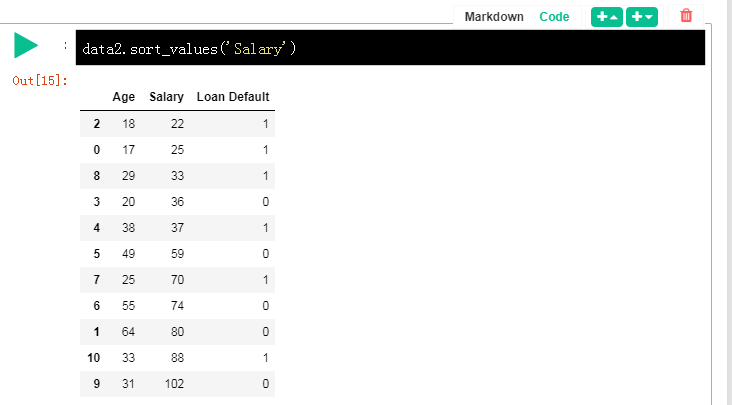
data2 = pd.DataFrame({'Age': [17, 64, 18, 20, 38, 49, 55, 25, 29, 31, 33], 'Salary': [25, 80, 22, 36, 37, 59, 74, 70, 33, 102, 88], 'Loan Default': [1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1]}) data2.sort\_values('Age')



上表可见，如果根据年龄排序，目标变量（Loan Default）将切换（从 1 到 0 或从 0 到 1）5 次。

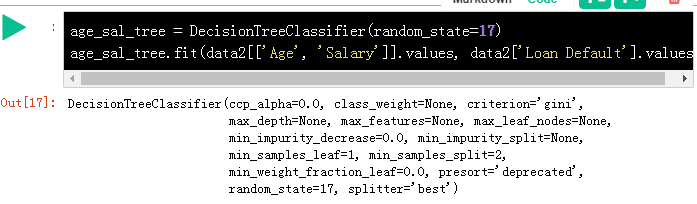
下表可见，如果根据薪水排序，它将切换 7 次

data2.sort\_values('Salary')



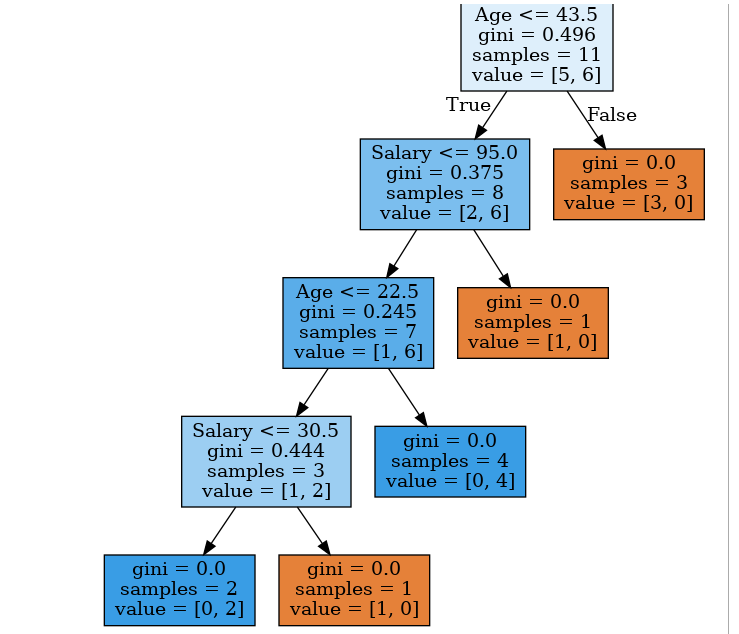
下面看看树将如何选择特征。

age\_sal\_tree = DecisionTreeClassifier(random\_state=17) age\_sal\_tree.fit(data2[['Age', 'Salary']].values, data2['Loan Default'].values)



dot\_data = StringIO() export\_graphviz(age\_sal\_tree, feature\_names=['Age', 'Salary'], out\_file=dot\_data, filled=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue()) Image(value=graph.create\_png())



上图表明，树同时根据薪水和年龄进行分区，有些节点的分割阈值选择了年龄，有些选择了薪水。树为何选择这些特征？因为根据基尼不确定性质量标准，它们提供了更好的分区。

结论：决策树处理数值特征最简单的启发式算法是升序排列它的值，然后只关注目标变量发生改变的那些值。

让我们基于以下函数生成一些带噪数据：

f(x) = e^{-x ^ 2} + 1.5 \* e^{-(x - 2) ^ 2}f(x)=e−x2+1.5∗e−(x−2)2

接着在生成的数据上训练一颗决策树，并进行预测，调用 plt 方法画出结果示意图。

from sklearn.tree import DecisionTreeRegressor n\_train = 150 n\_test = 1000 noise = 0.1

def f(x):

x = x.ravel()

return np.exp(-x \*\* 2) + 1.5 \* np.exp(-(x - 2) \*\* 2)

def generate(n\_samples, noise):

X = np.random.rand(n\_samples) \* 10 - 5

X = np.sort(X).ravel()

y = np.exp(-X \*\* 2) + 1.5 \* np.exp(-(X - 2) \*\* 2) + \ np.random.normal(0.0, noise, n\_samples)

X = X.reshape((n\_samples, 1))

return X, y

X\_train, y\_train = generate(n\_samples=n\_train, noise=noise)

X\_test, y\_test = generate(n\_samples=n\_test, noise=noise)

reg\_tree = DecisionTreeRegressor(max\_depth=5, random\_state=17) reg\_tree.fit(X\_train, y\_train)

reg\_tree\_pred = reg\_tree.predict(X\_test)

plt.figure(figsize=(10, 6))

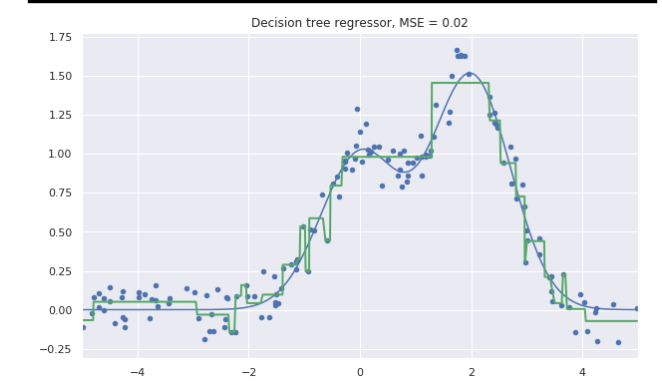
plt.plot(X\_test, f(X\_test), "b") plt.scatter(X\_train, y\_train, c="b", s=20)

plt.plot(X\_test, reg\_tree\_pred, "g", lw=2)

plt.xlim([-5, 5])

plt.title("Decision tree regressor, MSE = %.2f" % (np.sum((y\_test - reg\_tree\_pred) \*\* 2) / n\_test))

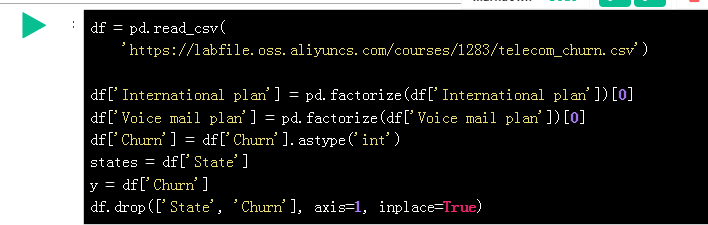
plt.show()



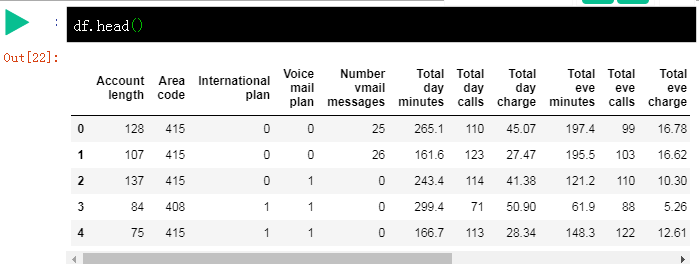
#### **在客户离网率预测任务中使用决策树和最近邻方法**

首先读取数据至 DataFrame 并进行预处理。将 State 特征从 DateFrame 转移到单独的 Series 对象中。我们训练的第一个模型将不包括 State 特征，之后再考察 State 特征是否有用。

df = pd.read\_csv( 'https://labfile.oss.aliyuncs.com/courses/1283/telecom\_churn.csv') df['International plan'] = pd.factorize(df['International plan'])[0] df['Voice mail plan'] = pd.factorize(df['Voice mail plan'])[0] df['Churn'] = df['Churn'].astype('int') states = df['State'] y = df['Churn'] df.drop(['State', 'Churn'], axis=1, inplace=True)



df.head()

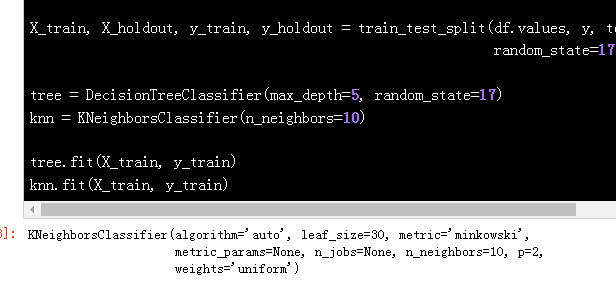


接下来，训练 2 个模型：决策树和 k-NN。一开始，我们并不知道如何设置模型参数能使模型表现好，所以可以使用随机参数方法，假定树深（max\_dept）为 5，近邻数量（n\_neighbors）为 10。

from sklearn.model\_selection

import train\_test\_split, StratifiedKFold from sklearn.neighbors

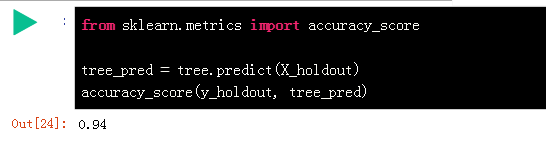
import KNeighborsClassifier X\_train, X\_holdout, y\_train, y\_holdout = train\_test\_split(df.values, y, test\_size=0.3, random\_state=17) tree = DecisionTreeClassifier(max\_depth=5, random\_state=17) knn = KNeighborsClassifier(n\_neighbors=10) tree.fit(X\_train, y\_train) knn.fit(X\_train, y\_train)



使用准确率（Accuracy）在留置集上评价模型预测的质量。

from sklearn.metrics

import accuracy\_score tree\_pred = tree.predict(X\_holdout) accuracy\_score(y\_holdout, tree\_pred)



knn\_pred = knn.predict(X\_holdout)

accuracy\_score(y\_holdout, knn\_pred)

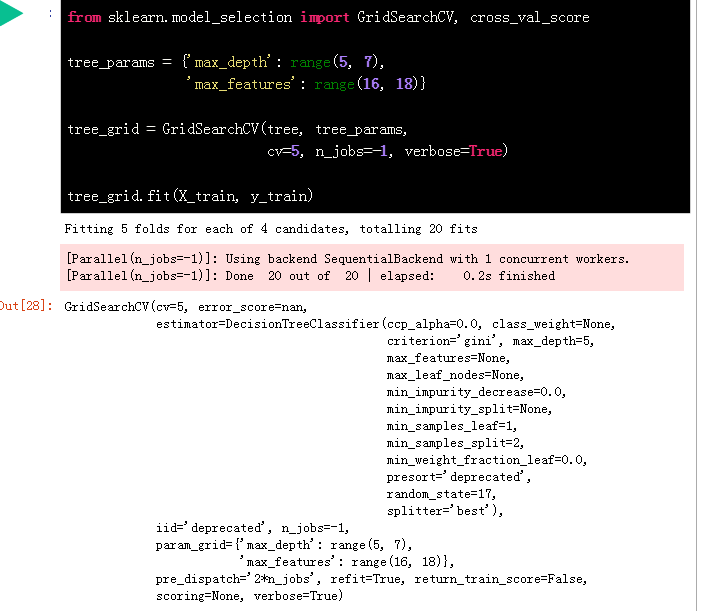


现在，使用交叉验证确定树的参数，对每次分割的 max\_dept（最大深度 h）和 max\_features（最大特征数）进行调优。GridSearchCV() 函数可以非常简单的实现交叉验证，下面程序对每一对 max\_depth 和 max\_features 的值使用 5 折验证计算模型的表现，接着选择参数的最佳组合

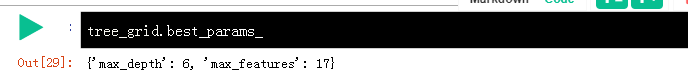
from sklearn.model\_selection

import GridSearchCV, cross\_val\_score tree\_params = {'max\_depth': range(5, 7), 'max\_features': range(16, 18)}

tree\_grid = GridSearchCV(tree, tree\_params, cv=5, n\_jobs=-1, verbose=True) tree\_grid.fit(X\_train, y\_train)



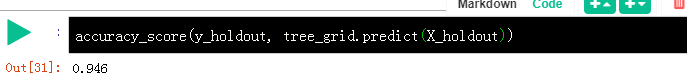
列出交叉验证得出的最佳参数和相应的训练集准确率均值。



tree\_grid.best\_score\_

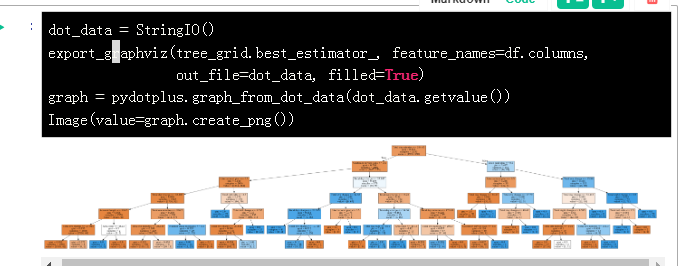


accuracy\_score(y\_holdout, tree\_grid.predict(X\_holdout))



绘制所得的决策树。

dot\_data = StringIO() export\_graphviz(tree\_grid.best\_estimator\_, feature\_names=df.columns, out\_file=dot\_data, filled=True) graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue()) Image(value=graph.create\_png())



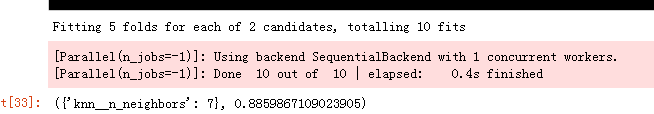
现在，再次使用交叉验证对 k-NN 的 k 值（即邻居数）进行调优。

from sklearn.pipeline

import Pipeline

from sklearn.preprocessing

import StandardScaler knn\_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n\_jobs=-1))]) knn\_params = {'knn\_\_n\_neighbors': range(6, 8)} knn\_grid = GridSearchCV(knn\_pipe, knn\_params, cv=5, n\_jobs=-1, verbose=True) knn\_grid.fit(X\_train, y\_train) knn\_grid.best\_params\_, knn\_grid.best\_score\_



knn\_grid.best\_params\_



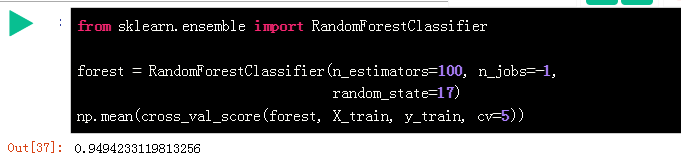
accuracy\_score(y\_holdout, knn\_grid.predict(X\_holdout))

0.89

使用 RandomForestClassifier() 方法再训练一个随机森林（可以把它想象成一群互相协作的决策树），看看能否在这个任务上有更好的表现。

from sklearn.ensemble

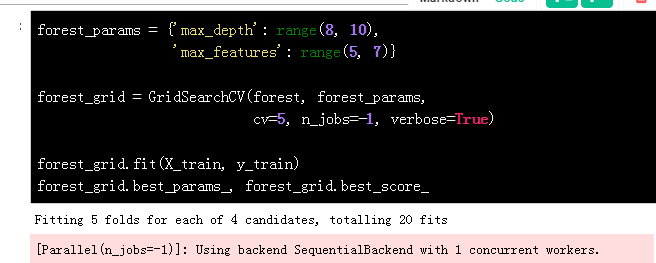
import RandomForestClassifier forest = RandomForestClassifier(n\_estimators=100, n\_jobs=-1, random\_state=17) np.mean(cross\_val\_score(forest, X\_train, y\_train, cv=5))



forest\_params = {'max\_depth': range(8, 10), 'max\_features': range(5, 7)} forest\_grid = GridSearchCV(forest, forest\_params, cv=5, n\_jobs=-1, verbose=True)

forest\_grid.fit(X\_train, y\_train)

forest\_grid.best\_params\_, forest\_grid.best\_score\_



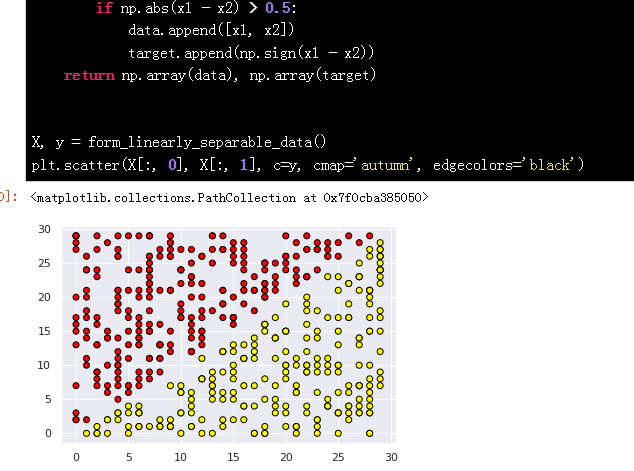
accuracy\_score(y\_holdout, forest\_grid.predict(X\_holdout))

0.953

**决策树的复杂情况**

1. 首先，在一个平面上创建一组具有 2 个分类的数据点，每个数据点是两个分类中的一个（红色表示 x\_1>x\_2x1​>x2​，黄色表示 x\_1<x\_2x1​<x2​），其实用一条直线 x\_1 = x\_2x1​=x2​ 就可以完成它们的分类，那么决策树会这么做吗？

def form\_linearly\_separable\_data(n=500, x1\_min=0, x1\_max=30, x2\_min=0, x2\_max=30): data, target = [], [] for i in range(n): x1 = np.random.randint(x1\_min, x1\_max) x2 = np.random.randint(x2\_min, x2\_max) if np.abs(x1 - x2) > 0.5: data.append([x1, x2]) target.append(np.sign(x1 - x2)) return np.array(data), np.array(target) X, y = form\_linearly\_separable\_data() plt.scatter(X[:, 0], X[:, 1], c=y, cmap='autumn', edgecolors='black')



1. 训练一个决策树对上面的数据进行分类，并绘制分类边界

tree = DecisionTreeClassifier(random\_state=17).fit(X, y) xx, yy = get\_grid(X) predicted = tree.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape) plt.pcolormesh(xx, yy, predicted, cmap='autumn') plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap='autumn', edgecolors='black', linewidth=1.5) plt.title('Easy task. Decision tree compexifies everything')

1. 可视化决策树。

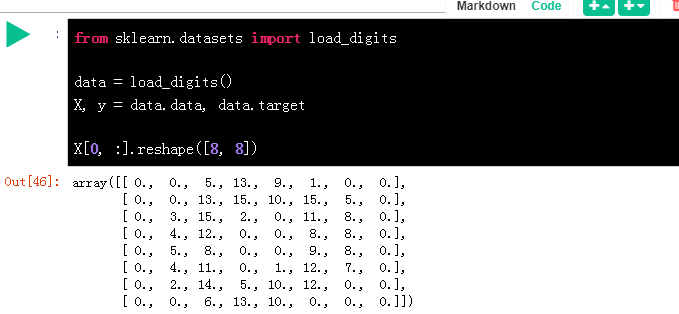
dot\_data = StringIO() export\_graphviz(tree, feature\_names=['x1', 'x2'], out\_file=dot\_data, filled=True) graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue()) Image(value=graph.create\_png())

**在 MNIST 手写数字识别任务中应用决策树和 k-NN**

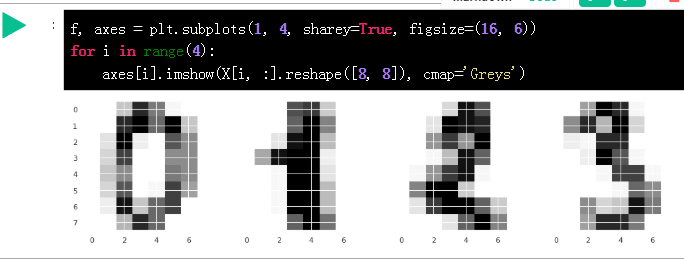
现在可以看看这两个算法应用到实际任务上的表现如何，首先载入 sklearn 内置的 MNIST 手写数字数据集，该数据库中手写数字的图片为 8x8 的矩阵，矩阵中的值表示每个像素的白色亮度。

from sklearn.datasets

import load\_digits data = load\_digits() X, y = data.data, data.target X[0, :].reshape([8, 8])



绘制一些 MNIST 手写数字。



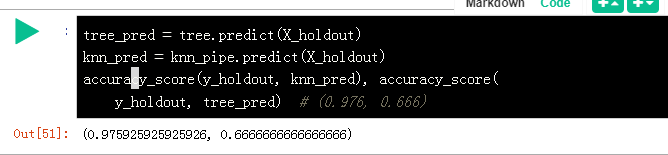
使用 train\_test\_split() 方法分割数据集，其中的 70% 作为训练集（X\_train，y\_train），30% 作为留置集（X\_holdout，y\_holdout）

X\_train, X\_holdout, y\_train, y\_holdout = train\_test\_split( X, y, test\_size=0.3, random\_state=17)

使用随机参数训练决策树和 k-NN。

tree = DecisionTreeClassifier(max\_depth=5, random\_state=17) knn\_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n\_neighbors=10))]) tree.fit(X\_train, y\_train) knn\_pipe.fit(X\_train, y\_train)

训练好之后，分别在留置集上做出预测。



从上可知，k-NN 做得更好，不过别忘了我们用的是随机参数。现在，使用交叉验证调优决策树模型，因为这次任务所需考虑的特征比之前任务中的更多，所以可以增加参数的大小。

tree\_params = {'max\_depth': [10, 20, 30], 'max\_features': [30, 50, 64]} tree\_grid = GridSearchCV(tree, tree\_params, cv=5, n\_jobs=-1, verbose=True) tree\_grid.fit(X\_train, y\_train)

查看交叉验证得到的最佳参数组合和相应的准确率。

tree\_grid.best\_params\_, tree\_grid.best\_score\_



从上可知，在这个数据集中随机森林的准确率（93.5%）不如 k-NN（98.7%）。当然，我们没有对随机森林的参数进行任何调优，但即使经过调优，训练精确度也无法超过 k-NN。

**决策树和最近邻方法的优势和劣势**

优势：

* 生成容易理解的分类规则，这一属性称为模型的可解释性。例如它生成的规则可能是「如果年龄不满 25 岁，并对摩托车感兴趣，那么就拒绝发放贷款」。
* 很容易可视化，即模型本身（树）和特定测试对象的预测（穿过树的路径）可以「被解释」。
* 训练和预测的速度快。
* 较少的参数数目。
* 支持数值和类别特征。

劣势：

* 决策树对输入数据中的噪声非常敏感，这削弱了模型的可解释性。
* 决策树构建的边界有其局限性：它由垂直于其中一个坐标轴的超平面组成，在实践中比其他方法的效果要差。
* 我们需要通过剪枝、设定叶节点的最小样本数、设定树的最大深度等方法避免过拟合。
* 不稳定性，数据的细微变动都会显著改变决策树。这一问题可通过决策树集成方法来处理（以后的实验会介绍）。
* 搜索最佳决策树是一个「NP 完全」（NP-Complete）问题。了解什么是 NP-Complete 请点击 [这里](https://baike.baidu.com/item/NP-Complete/15961931?fr=aladdin" \t "https://www.lanqiao.cn/courses/2325/learning/_blank)。实践中使用的一些推断方法，比如基于最大信息增益进行贪婪搜索，并不能保证找到全局最优决策树。
* 倘若数据中出现缺失值，将难以创建决策树模型。Friedman 的 CART 算法中大约 50% 的代码是为了处理数据中的缺失值（现在 sklearn 实现了这一算法的改进版本）。
* 这一模型只能内插，不能外推（随机森林和树提升方法也是如此）。也就是说，倘若你预测的对象在训练集所设置的特征空间之外，那么决策树就只能做出常数预测。比如，在我们的黄球和蓝球的例子中，这意味着模型将对所有位于 >19 或 <0 的球做出同样的预测。

**最近邻方法**

优势：

* 实现简单。
* 研究很充分。
* 通常而言，在分类、回归、推荐问题中第一个值得尝试的方法就是最近邻方法。
* 通过选择恰当的衡量标准或核，它可以适应某一特定问题。
* 劣势：
* 和其他复合算法相比，这一方法速度较快。但是，现实生活中，用于分类的邻居数目通常较大（100-150），在这一情形下，k-NN 不如决策树快。
* 如果数据集有很多变量，很难找到合适的权重，也很难判定哪些特征对分类/回归不重要。
* 依赖于对象之间的距离度量，默认选项欧几里得距离常常是不合理的。你可以通过网格搜索参数得到良好的解，但在大型数据集上的耗时很长。
* 没有理论来指导我们如何选择邻居数，故而只能进行网格搜索（尽管基本上所有的模型，在对其超参数进行调整时都使用网格搜索的方法）。在邻居数较小的情形下，该方法对离散值很敏感，也就是说，有过拟合的倾向。
* 由于「维度的诅咒」，当数据集存在很多特征时它的表现不佳。

1. ****实验总结****

本次实验中，决策树完美地解决了问题，在交叉验证和留置集上都得到了 100% 的准确率。其实，k-NN 之所以在这个任务上表现不佳并非该方法本身的问题，而是因为使用了欧几里得距离，因为欧几里得距离没能察觉出有一个特征（成比例）比其他所有特征（噪声）更重要。

1. ****心得体会****

本次实验通过决策树和最近邻方法在几个简单示例上构建了分类模型，并基于交叉验证的方法对模型进行调优，之后对比了决策树和最近邻方法的优劣情况。这两种方法常常作为机器学习模型的基线，如果想要学好机器学习，学好它们是非常重要的。