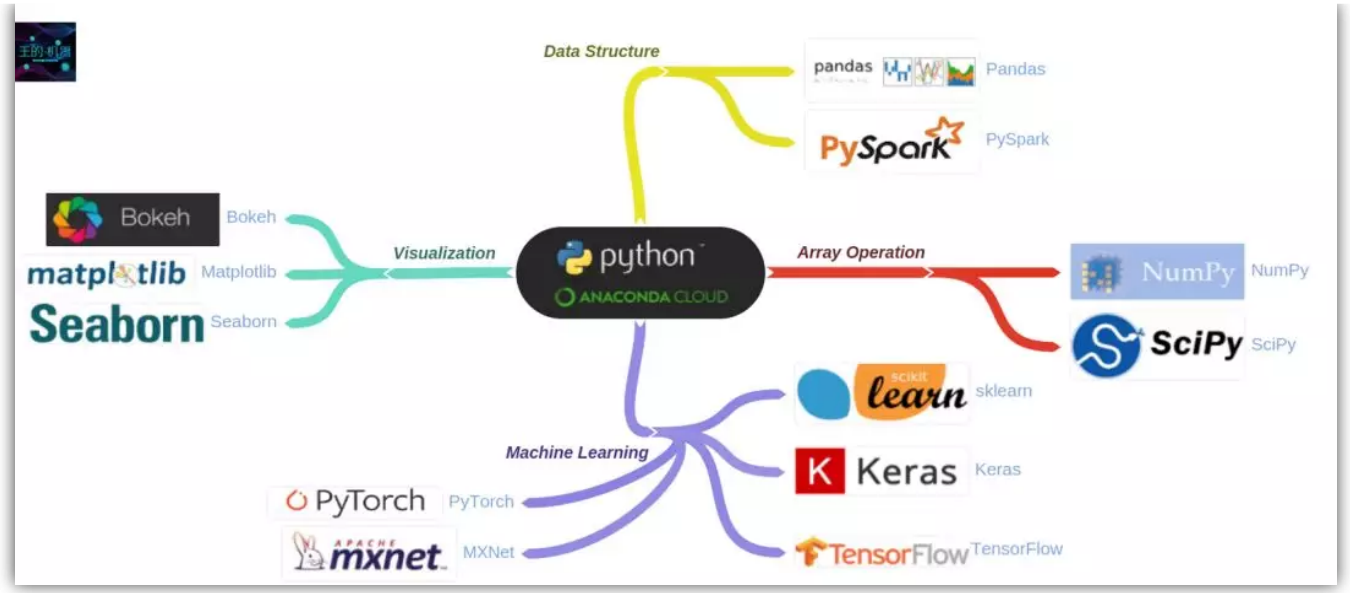
盘一盘 Python 系列特别篇 - Sklearn (0.22)

原创 王圣元 王的机器 2019-12-14

来自专辑

Python



本文含 5199 字，33 图表截屏

建议阅读 26 分钟

0 引言

本文是 Python 系列的特别篇的第五篇

- **特别篇 1 - PyEcharts TreeMap**
- **特别篇 2 - 面向对象编程**
- **特别篇 3 - 两大利「器」**
- **特别篇 4 - 装饰器**
- **特别篇 5 - Sklearn 0.22**

在〖**机器学习之 Sklearn**〗一贴中，我们已经介绍过 Sklearn，它全称是 Scikit-learn，是基于 Python 语言的机器学习工具。

在 2019 年 12 月 3 日，Sklearn 已经更新到版本 0.22，里面添加了若干功能，这也是本帖的内容。

首先在 Anaconda 的提示窗中输入以下代码，来安装更新当前的 Sklearn，如果代码运行时报错就以管理员身份打开提示窗。

```
pip install --upgrade scikit-learn
```

检查一下 Sklearn 的版本确定已经更新到 0.22。

```
import sklearn
print( sklearn.__version__ )
```

```
0.22
```

在添加的众多功能中，我觉得以下几个算是比较有用的。

- 一行画出 ROC-AUC 图
- 实现堆积法 (stacking)
- 为任何模型估计特征重要性
- 用 k-近邻法来填充缺失值

首先加载下面例子共用的包。

```
import matplotlib.pyplot as plt
%matplotlib inline
seed = 1031

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

1 ROC-AUC

首先介绍一下接受者操作特征 (ROC)。

接受者操作特征

ROC 是 receiver operating characteristic 的简称，直译为「接受者操作特征」。「ROC 曲线」非常类似「PR 曲线」，但图的横轴纵轴并不是查准率和查全率。「ROC 曲线」反映在不同分类阈值上，真正类率 (true positive rate, TPR) 和假正类率 (false positive rate, FPR) 的关系。

- TPR 是「真正类」和所有正类 (真正类+假负类) 的比率，真正类率 = 查全率
- FPR 是「假正类」和所有负类 (假正类+真负类) 的比率，假正类率 = 1- 真负类率 = 1 - 特异率 (specificity)

一般来说，阈值越高

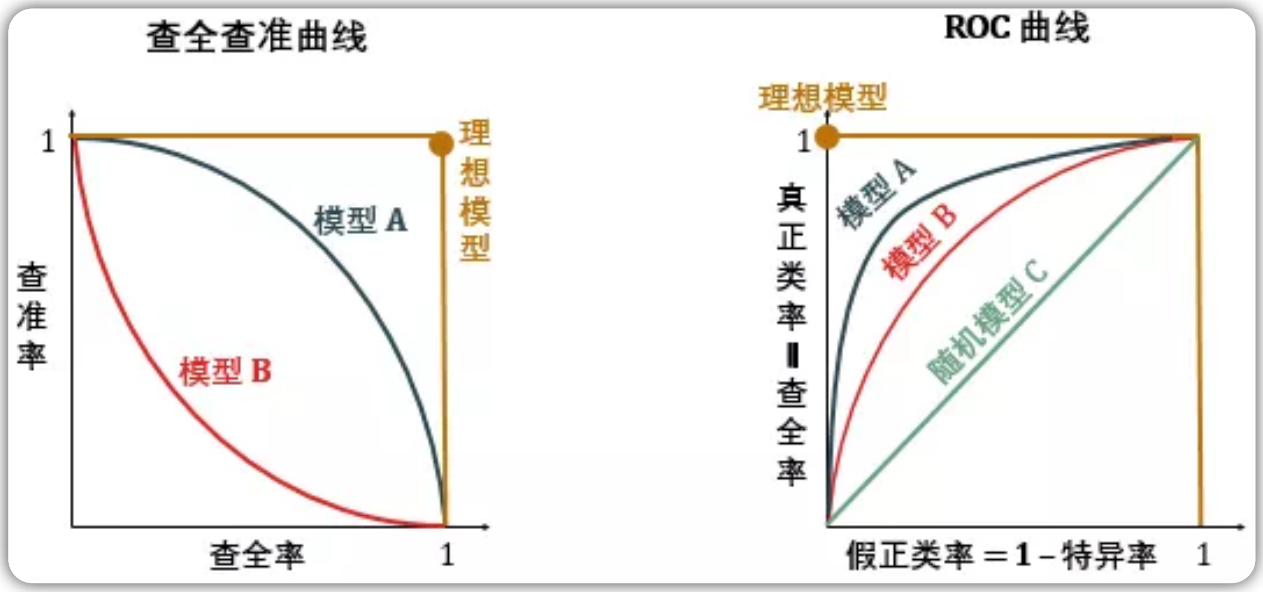
- 越不容易预测出正类，TPR **下降**（TPR 和阈值成**递减**关系)
- 越容易预测出负类，(1- FPR) **上升**（FPR 和阈值成**递减**关系)

阈值越低

- 越容易预测出正类，TPR **上升**（TPR 和阈值成**递减**关系)
- 越不容易预测出负类，(1- FPR) **下降**（FPR 和阈值成**递减**关系)

因此 TPR 和 FPR 是单调递增关系。

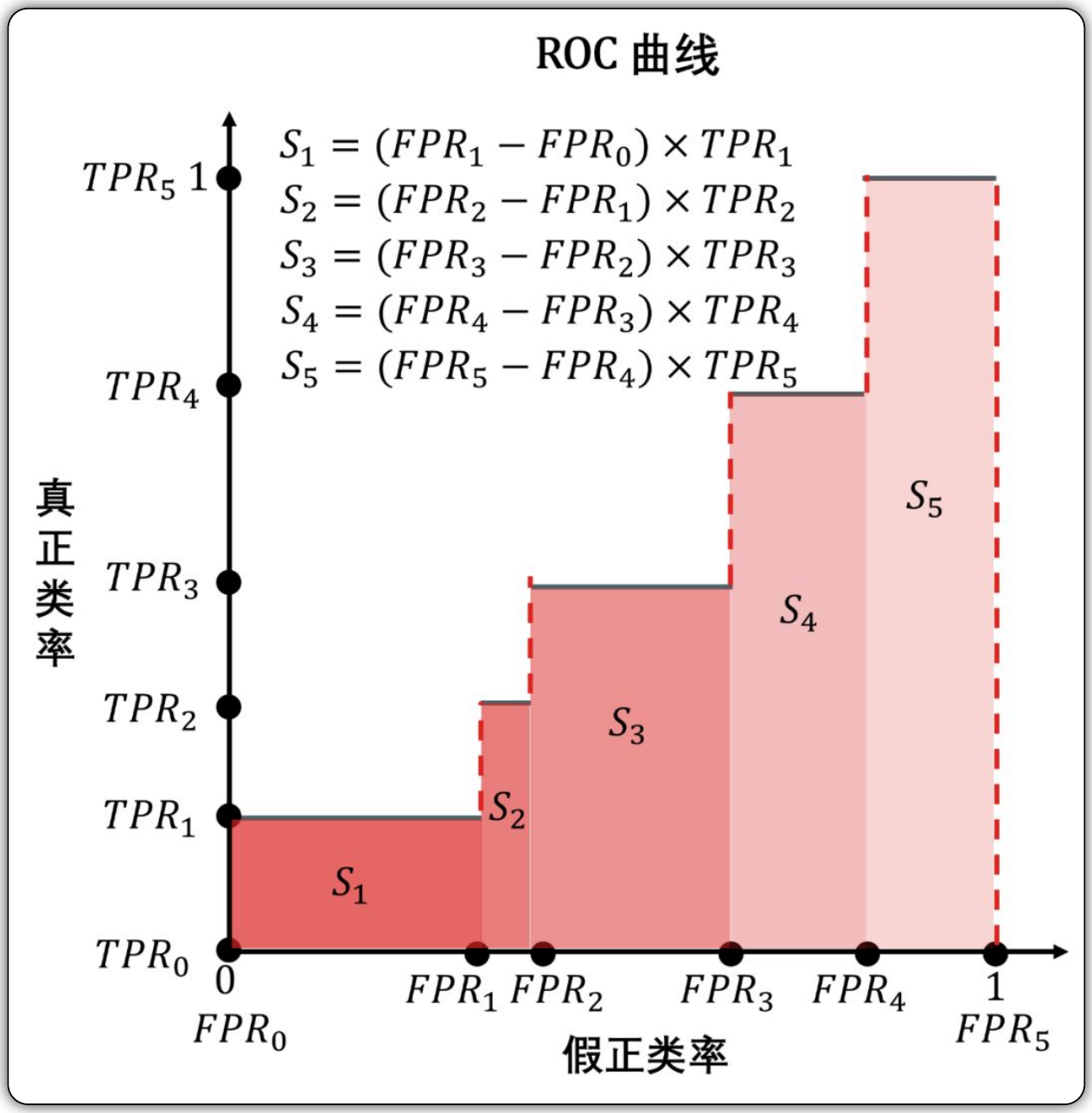
「PR 曲线」和「ROC 曲线」对比图见下，后者和横轴之间的面积叫AUC，是 area under the curve 的简称。



AUC 将所有可能分类阈值的评估标准浓缩成一个数值，根据 AUC 大小，我们得出

$$\overbrace{\text{理想模型}}^{\text{面积为 1}} > \text{模型 A} > \text{模型 B} > \overbrace{\text{随机模型}}^{\text{面积为 0.5}}$$

如何计算 AUC 和计算 PR 曲线下的面积一样的，把横坐标和纵坐标代表的变量弄对就可以了，如下图。



如何确定这些 TPR_i 和 FPR_i ($i = 0,1,\ldots,5$) 不是一件容易讲清的事，我试试，先看一个二分类预测类别以及预测正类概率的表 (按照预测概率降序排序，其中正类 P 和负类 N 都有 10 个)。

样本	类别	预测概率	样本	类别	预测概率
1	P	0.9	11	P	0.4
2	P	0.8	12	N	0.39
3	N	0.7	13	P	0.38
4	P	0.6	14	N	0.37
5	P	0.55	15	N	0.36
6	P	0.54	16	N	0.35
7	N	0.53	17	P	0.34
8	N	0.52	18	N	0.33
9	P	0.51	19	P	0.3
10	N	0.505	20	N	0.1

第一个点：当阈值 = 0.9，那么第 1 个样本预测为 P，后 19 个样本预测为 N，这时

- TPR = 真正类/全部正类 = 1/10 =0.1
- FPR = 1 - 真负类/全部负类 = 1 - 10/10 =0
- 阈值 0.9 → (0, 0.1)

第二个点：当阈值 = 0.8，那么第 1, 2 个样本预测为 P，后 18 个样本预测为 N，这时

- TPR = 真正类/全部正类 = 2/10 =0.2
- FPR = 1 - 真负类/全部负类 = 1 - 10/10 =0
- 阈值 0.8 → (0, 0.2)

...

第四个点：当阈值 = 0.6，那么前 4 个样本预测为 P，后 16 个样本预测为 N，这时

- TPR = 真正类/全部正类 = 3/10 =0.3
- FPR = 1 - 真负类/全部负类 = 1 - 9/10 =0.1
- 阈值 0.8 → (0.1, 0.3)

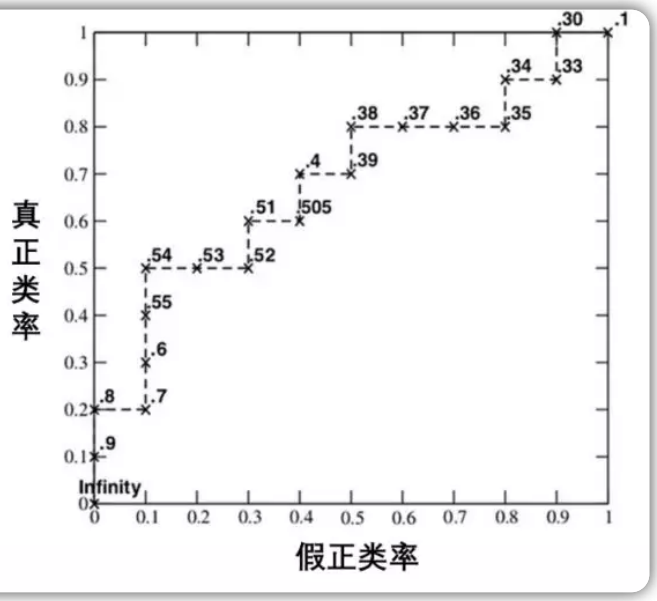
...

最后一个点：当阈值 = 0.1，那么全部样本预测为 P，零样本预测为 N，这时

- TPR = 真正类/全部正类 = 10/10 =1
- FPR = 1 - 真负类/全部负类 = 1 - 0/10 =1
- 阈值 0.8 → (1, 1)

因此可画出下图右半部分，即 ROC 曲线，再根据横坐标纵坐标上的 FPR 和 TPR 计算 AUC。

样本	类别	预测概率	样本	类别	预测概率
1	P	0.9	11	P	0.4
2	P	0.8	12	N	0.39
3	N	0.7	13	P	0.38
4	P	0.6	14	N	0.37
5	P	0.55	15	N	0.36
6	P	0.54	16	N	0.35
7	N	0.53	17	P	0.34
8	N	0.52	18	N	0.33
9	P	0.51	19	P	0.3
10	N	0.505	20	N	0.1



AUC 越大，分类器的质量越好。

在 Scikit-learn 里，还记得有三种方式引入数据吗？

1. 用 `load_dataname` 来加载小数据
2. 用 `fetch_dataname` 来下载大数据
3. 用 `make_dataname` 来构造随机数据

这里我们用第三种：

```
X, y = datasets.make_classification(random_state=seed)
X_train, X_test, y_train, y_test
= train_test_split(X, y, random_state=seed)
```

用支持向量机分类器 svc 和随机森林分类器 rfc 来训练一下。

```
svc = SVC()
svc.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

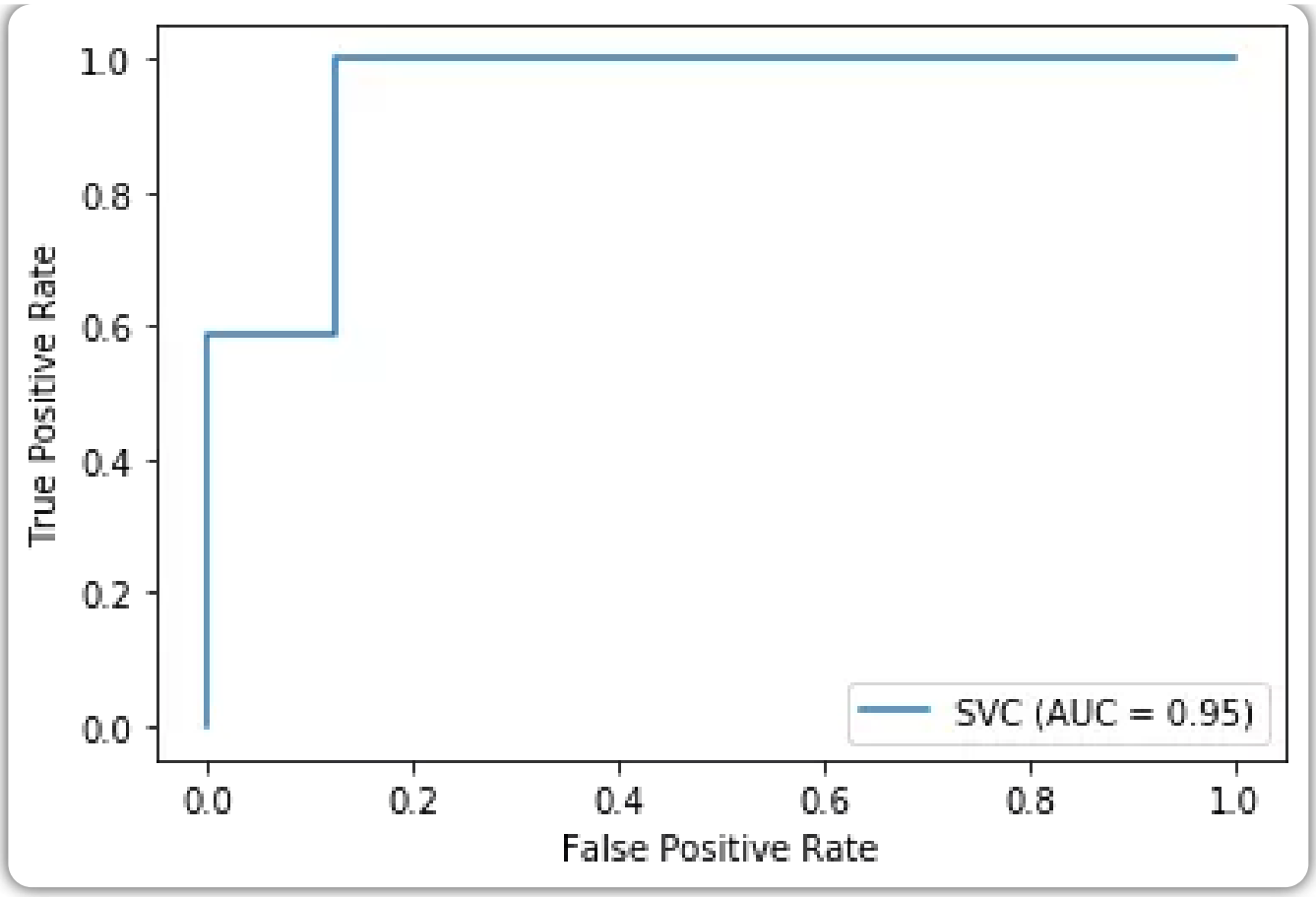
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

一行画 ROC 图需要引入 `plot_roc_curve` 包。

```
from sklearn.metrics import plot_roc_curve
```

再运行下面一行代码，需要传进三个参数：估计器 `svc`，特征 `x_test`，标签 `y_test`。

```
plot_roc_curve( svc, X_test, y_test );
```

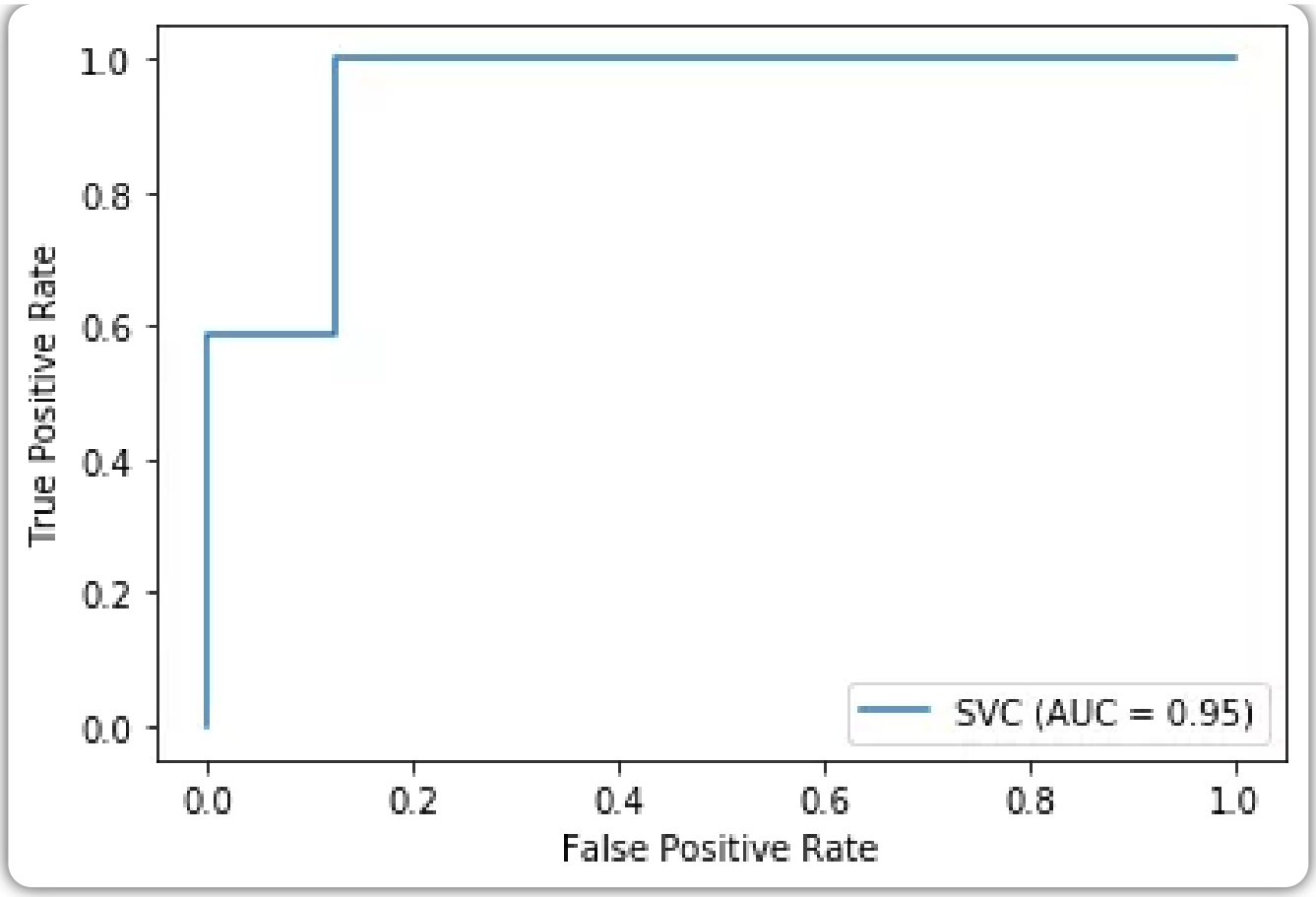


在版本 v0.22 之前，要画出上面这图需要写好多行代码：

```
from sklearn.metrics import roc_curve, auc
```

```
y_score = svc.decision_function(X_test)
fpr, tpr, _ = roc_curve( y_test, y_score )
roc_auc = auc(fpr, tpr)
```

```
plt.figure()
plt.plot(fpr, tpr, label='SVC (AUC = %0.2f)' % roc_auc)
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```



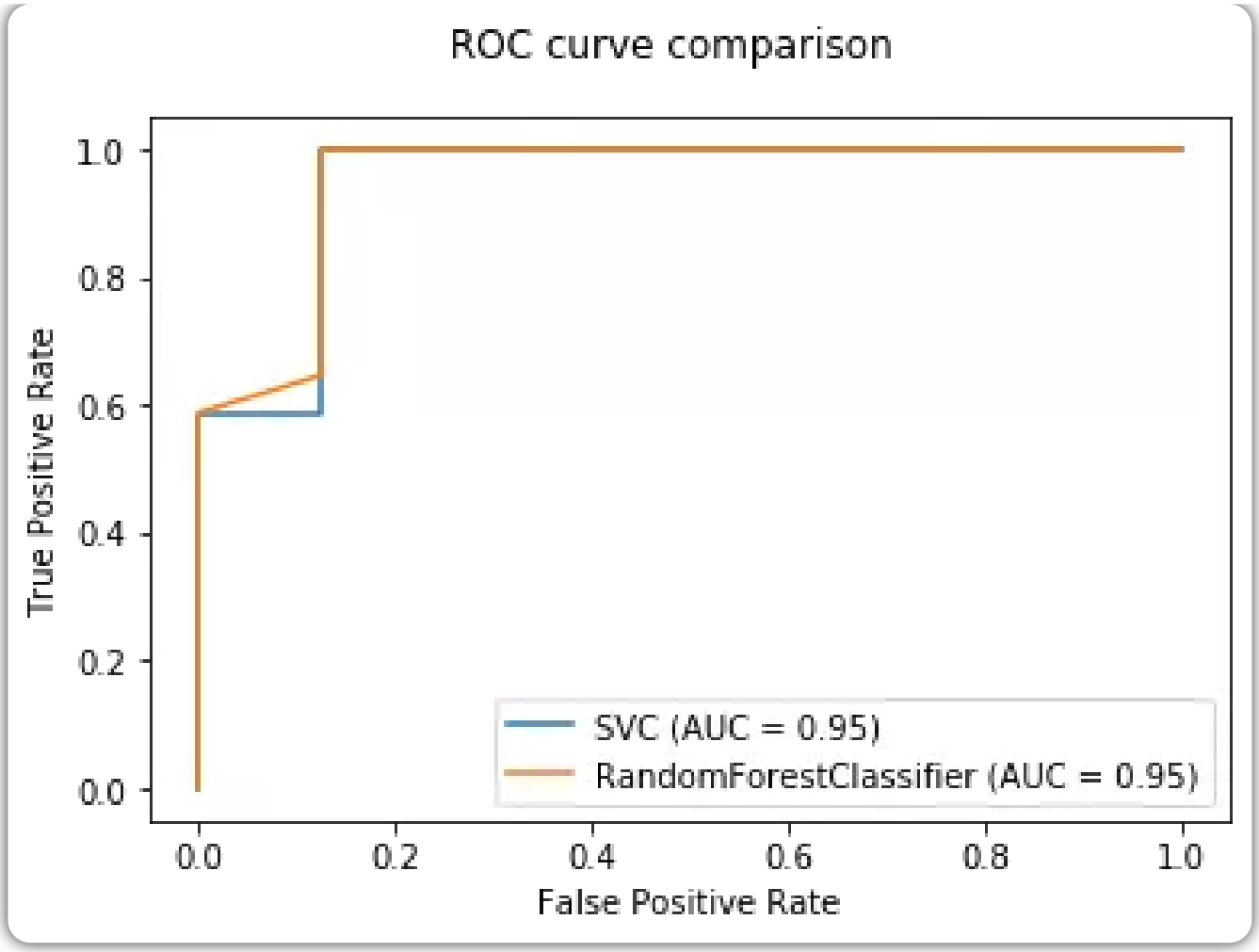
所以现在这个 `plot_roc_curve` 包真的方便，不过其实在 Scikit-plot 里，早就有类似的函数了，见〖**机学可视化之 Scikit-Plot**〗一贴。那个函数叫做 `plot_roc` 用到的参数有 3 个：

- `y_test`：测试集真实标签
- `y_prob`：测试集预测标签的概率
- `figsize`：图片大小

我猜想 v0.22 是借鉴了 Scikit-plot 里 `plot_roc` 函数的写法得到了 `plot_roc_curve`。

此外，`plot_roc_curve` 函数还可以画出不同估计器得到的 ROC 曲线。只需要将 svc 模型下的 ROC 图中的坐标系传到 rfc 模型下的 ROC 图中的 `ax` 参数中。代码如下：

```
svc_disp = plot_roc_curve(svc, X_test, y_test)
rfc_disp = plot_roc_curve(rfc, X_test, y_test, ax=svc_disp.ax_)
rfc_disp.figure_.suptitle("ROC curve comparison")

plt.show()
```

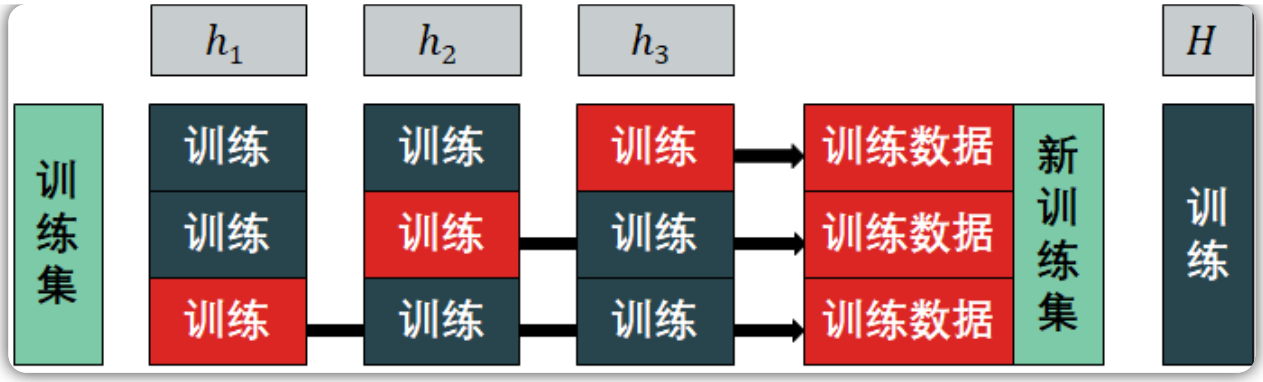


2 Stacking

首先介绍一下堆积法 (stacking)。

堆积法

堆积法实际上借用交叉验证思想来训练一级分类器，解释如下图：



训练一级分类器 – 首先将训练数据分为 3 份：D_1, D_2, D_3，h_1 在 D_1 和 D_2 上训练，h_2 在 D_1 和 D_3 上训练，h_3 在 D_2 和 D_3 上训练。

新训练数据 – 包含：h_1 在 D_3 上的产出，h_2 在 D_2 上的产出，h_3 在 D_1 上的产出。

训练二级分类器 – 在新训练数据和对应的标签上训练出第二级分类器 H。

接着我们拿**手写数字** (MNIST) 数据举例。

```
from sklearn.datasets import fetch_openml
```

下面也是 v0.22 的一个特功能 (但我觉得没什么太大用)：可以从 openML 返回数据帧的值，需要将 `as_frame` 参数设置为 `True`。再用操作符 `.` 来获取 X 和 y。，代码如下：

```
X_y = fetch_openml( 'mnist_784',
                    version=1,
                    as_frame=True )
X, y = X_y.data, X_y.target
```

其实我觉得原来将 `return_X_y` 参数设置为 `True`，一次性的把 X 和 y 都返回出来更简便，代码如下：

```
X, y = fetch_openml( 'mnist_784',
                     version=1,
                     return_X_y=True )
```

按 80:20 划分训练集和测试集，在标准化照片像素使得值在 0-1 之间。

```
X_train, X_test, y_train, y_test = \
train_test_split( X, y, test_size=0.2, random_state=seed )

X_train, X_test = X_train/255.0, X_test/255.0

print( 'The size of X_train is ', X_train.shape )
print( 'The size of y_train is ', y_train.shape )
print( 'The size of X_test is ', X_test.shape )
print( 'The size of y_test is ', y_test.shape )
```

```
The size of X_train is  (56000, 784)
The size of y_train is  (56000,)
The size of X_test is  (14000, 784)
The size of y_test is  (14000,)
```

接下来重头戏来了，用 StackingClassifier 作为元估计器（meta-estimators），来集成两个子估计器（base-estimator），我们用了**随机森林分类器** rfc 和**梯度提升分类器** gbc 作为一级分类器，之后用对率回归分类器作为二级分类器。

代码非常简单，如下：

```
from sklearn.ensemble import StackingClassifier

rfc = RandomForestClassifier(n_estimators=10, n_jobs=-1)
gbc = GradientBoostingClassifier(n_estimators=10)

estimators = [
    ('rf', rfc),
    ('gbt', gbc)
]

clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression()
)
```

代码通用格式为 (其中 FE 代表一级估计器，BE 代表，SE 代表)

```
FE  = [('BE1', BE1),
       ('BE2', BE2),
           ...,
       ('BEn', BEn)]

clf =  StackingClassifier(
           estimators = FE
```

```
                    final_estimators = SE)
```

在 Scikit-learn 里没有实现 `StackingClassifier`，我们只能用 mlxtend 里面的模型。现在在版本 0.22 里不需要这么做了。

```
from mlxtend.classifier import StackingClassifier
```

比较**子估计器**和**元估计器**的在测试集上的表现。

```
rfc.fit(X_train, y_train)
gbc.fit(X_train, y_train)
clf.fit(X_train, y_train)
rfc.score(X_test, y_test)
gbc.score(X_test, y_test)
clc.score(X_test, y_test)
```

```
0.9482142857142857
0.8391428571428572
1.0
```

集成分类器的得分比随机森林分类器和梯度提升分类器都高，几乎完全分类正确测试集了。堆积法的效果还真不错。

3 Feature Importance

首先介绍一下如何用置换检验 (permutation test) 来计算特征重要性 (feature importance)。

置换检验计算特征重要性

核心思想是“**如果某个特征是重要特征，那么加入一些随机噪声模型性能会下降**”。

做法是把所有数据在特征上的值重新随机排列，此做法被称为置换检验。这样可以保证随机打乱的数据分布和原数据接近一致。下图展示了在特征“性格”上随机排列后的数据样貌，随机排列将“好坏坏好坏坏好好”排成“坏坏好坏好坏坏好”。

自助采样好的数据

	长相	性格	收入	见吗?
1	好看	好	50万	是
2	一般	坏	44万	否
2	一般	坏	44万	否
8	难看	好	27万	是
6	一般	坏	3万	否
5	一般	坏	6万	否
7	难看	好	33万	是
4	好看	坏	11万	是

在特征性格上做随机排列

置换之后的数据

	长相	性格	收入	见吗?
1	好看	坏	50万	是
2	一般	坏	44万	否
2	一般	好	44万	否
8	难看	坏	27万	是
6	一般	好	3万	否
5	一般	坏	6万	否
7	难看	坏	33万	是
4	好看	好	11万	是

在置换检验后，特征的重要性可看成是模型“在原数据的性能”和“在特征数据置换后的性能”的差距，有

$$\text{特征重要性} = \left| \text{性能}\left(\overset{\text{原有数据}}{\hat{D}}\right) - \text{性能}\left(\overset{\text{置换之后数据}}{\hat{D}^{P}}\right) \right|$$

接着我们拿**鸢尾花** (iris) 数据举例。

首先按 80:20 划分训练集和测试集。

```
from sklearn.inspection import permutation_importance
from sklearn.datasets import load_iris

iris = load_iris()
X_train, X_test, y_train, y_test \
= train_test_split( iris['data'], iris['target'],
                    test_size=0.2, random_state=seed )
```

用随机森林训练，并在每个特征维度上做 10 次置换操作，注意 `n_repeats` 设置为 10。

```
rf = RandomForestClassifier().fit(X_train, y_train)
result = permutation_importance( rf, X_train, y_train,
                                 n_repeats=10, n_jobs=-1 )
```

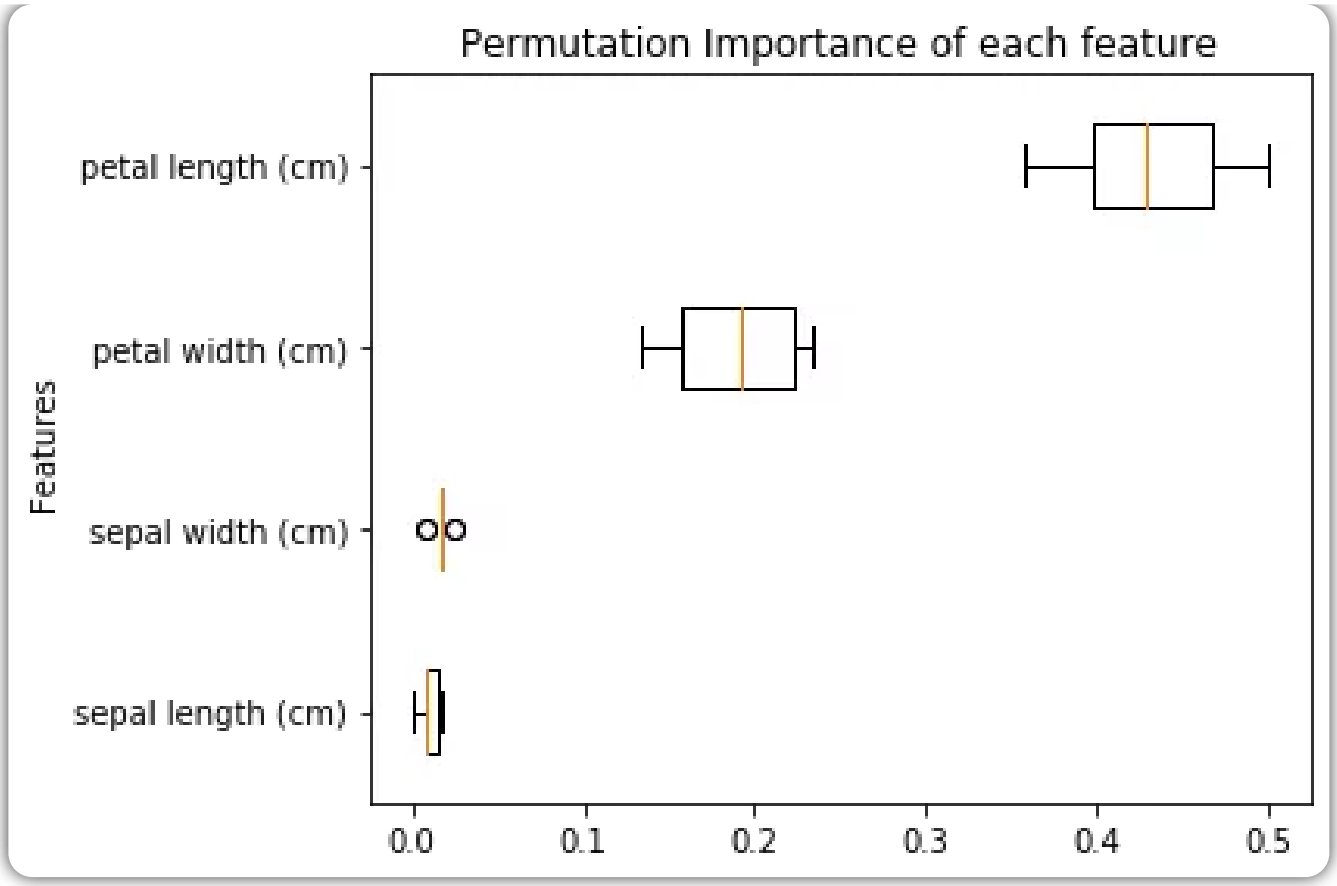
打印出 `result`，得到重要性的**均值**、**标准差**和 **10 组具体值**。

```
result
```

```
{'importances_mean': array([0.00916667, 0.01666667, 0.43166667, 0.19      ]),
 'importances_std': array([0.00583333, 0.00527046, 0.04711098, 0.03411582]),
 'importances': array([[0.00833333, 0.        , 0.00833333, 0.01666667, 0.
         0.01666667, 0.00833333, 0.00833333, 0.00833333, 0.01666667],
        [0.025     , 0.00833333, 0.025     , 0.01666667, 0.01666667,
         0.01666667, 0.01666667, 0.00833333, 0.01666667, 0.01666667],
        [0.425     , 0.49166667, 0.5       , 0.43333333, 0.35833333,
         0.39166667, 0.46666667, 0.46666667, 0.41666667, 0.36666667],
        [0.18333333, 0.225     , 0.23333333, 0.15      , 0.13333333,
         0.18333333, 0.21666667, 0.2       , 0.225     , 0.15      ]])}
```

这种数据形式最适合用箱形图 (box plot) 展示，**均值**是用来决定哪个特征最重要的，在箱形图中用一条线表示 (通常这条线指的中位数)。

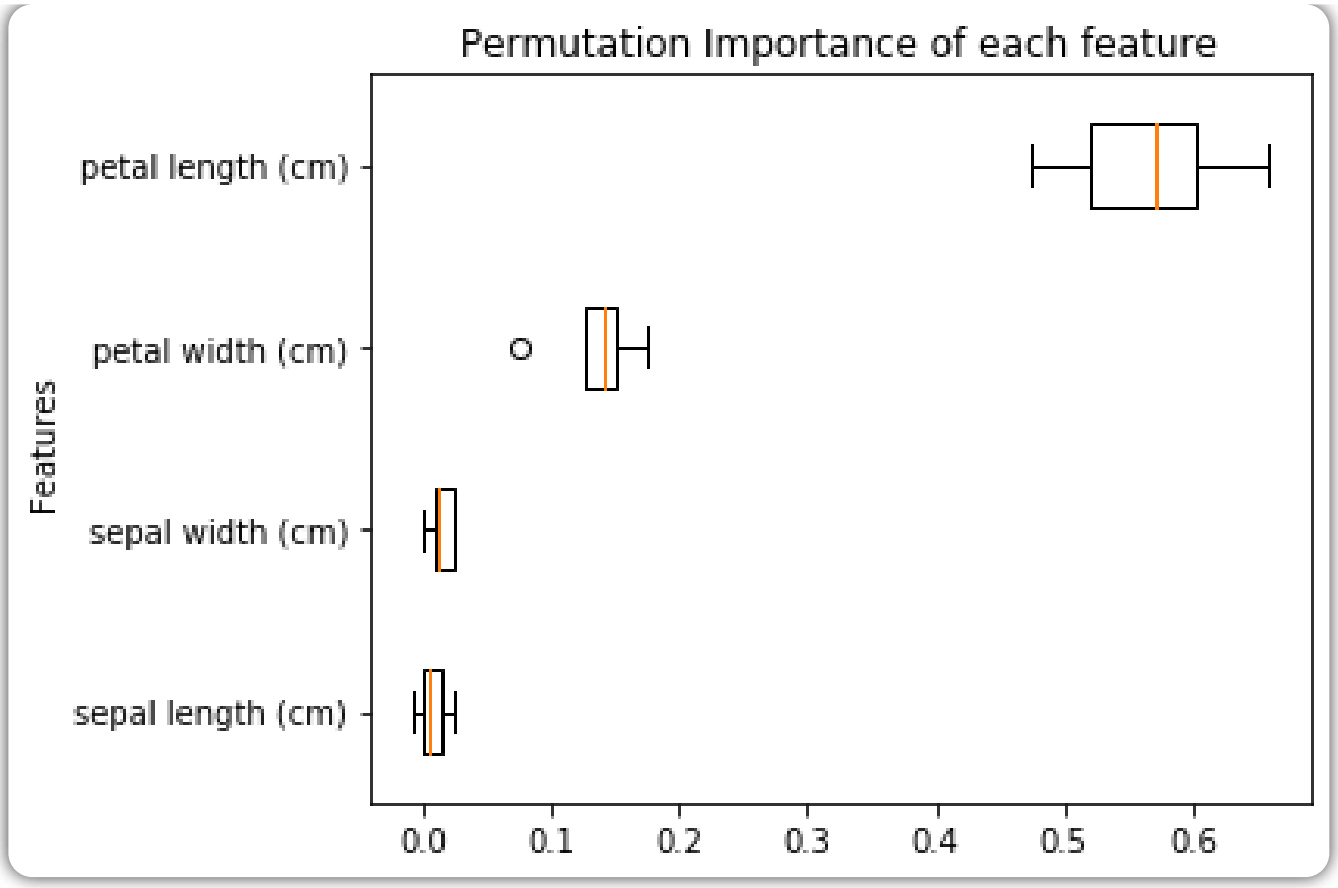
```
fig, ax = plt.subplots()
sorted_idx = result.importances_mean.argsort()
ax.boxplot( result.importances[sorted_idx].T,
            vert=False, labels=np.array(features)[sorted_idx] )
ax.set_title("Permutation Importance of each feature")
ax.set_ylabel("Features")
fig.tight_layout()
plt.show()
```



根据上图，我们得出**花瓣长度** (petal length) 特征最重要，而**花萼长度** (sepal length) 特征最不重要。当特征多时可用该方法来选择特征。

这个 `permutation_importance` 函数可以用在任何估计器上，再用一个支持向量机 svc。

```
svc = SVC().fit(X_train, y_train)
result = permutation_importance( svc, X_train, y_train,
                                 n_repeats=10, n_jobs=-1 )
```



根据上图，我们得出同样结论，**花瓣长度**特征最重要，**花萼长度**特征最不重要，虽然具体特征重要性均值和标准差不同，但在判断特征重要性的大方向还是一致的。

4 KNN Imputation

缺失数据的处理方式通常有三种：删除(delete)、推算(impute)和归类(categorize)。下面举例用的数据如下：

长相	性格	收入	见吗?
好看	好	50万	是
一般	?	40万	否
一般	好	30万	是
难看	坏	5万	否
好看	坏	7万	否
一般	坏	10万	是
难看	坏	9万	否
好看	坏	?	是
一般	?	12万	是

删除法

删除数据最简单，有两种方式：

- 删除行 (数据点)
- 删除列 (特征)

删除法的**优点**是

- 操作简单
- 可以用在任何模型比如决策树、线性回归等等

删除法的**缺点**是

- 删除的数据可能包含重要信息
- 不知道删除行好还是删除列好
- 对缺失数据的测试集没用

推算法

根据特征值是分类型或数值变量，两种方式：

1. 用众数来推算分类型
2. 用平均数来推算数值

特征“性格”的特征值是个分类型变量，因此计数未缺失数据得到 2 个好和 7 个坏，根据众数原则应该将缺失数据用“坏”来填充。特征“收入”的特征值是个数值型变量，根据平均数原则算出未缺失数据的均值 20.4 万来填充。

推算法的**优点**是

- 操作简单
- 可以用在任何模型比如决策树和线性回归等等
- 对缺失数据的测试集有用，运用同样的规则 (众数分类型变量，平均数数值型变量)

推算法的**缺点**是可能会造成系统型误差。

系统性误差有真实案例。在华盛顿的银行里申请贷款，根据当地法律，申请人是不允许填年龄的。如果整合所有美国申请人资料，发现所有来自华盛顿的数据缺失年龄那一栏。假如按照“平均化数值型变量”规则算出均值 41岁，那么把所有华盛顿申请者的年龄填为 41 岁是不合理的。

长相	性格	收入	见吗?
好看	好	50万	是
一般	?	40万	否
一般	好	30万	是
难看	坏	5万	否
好看	坏	7万	否
一般	坏	10万	是
难看	坏	9万	否
好看	坏	?	是
一般	?	12万	是

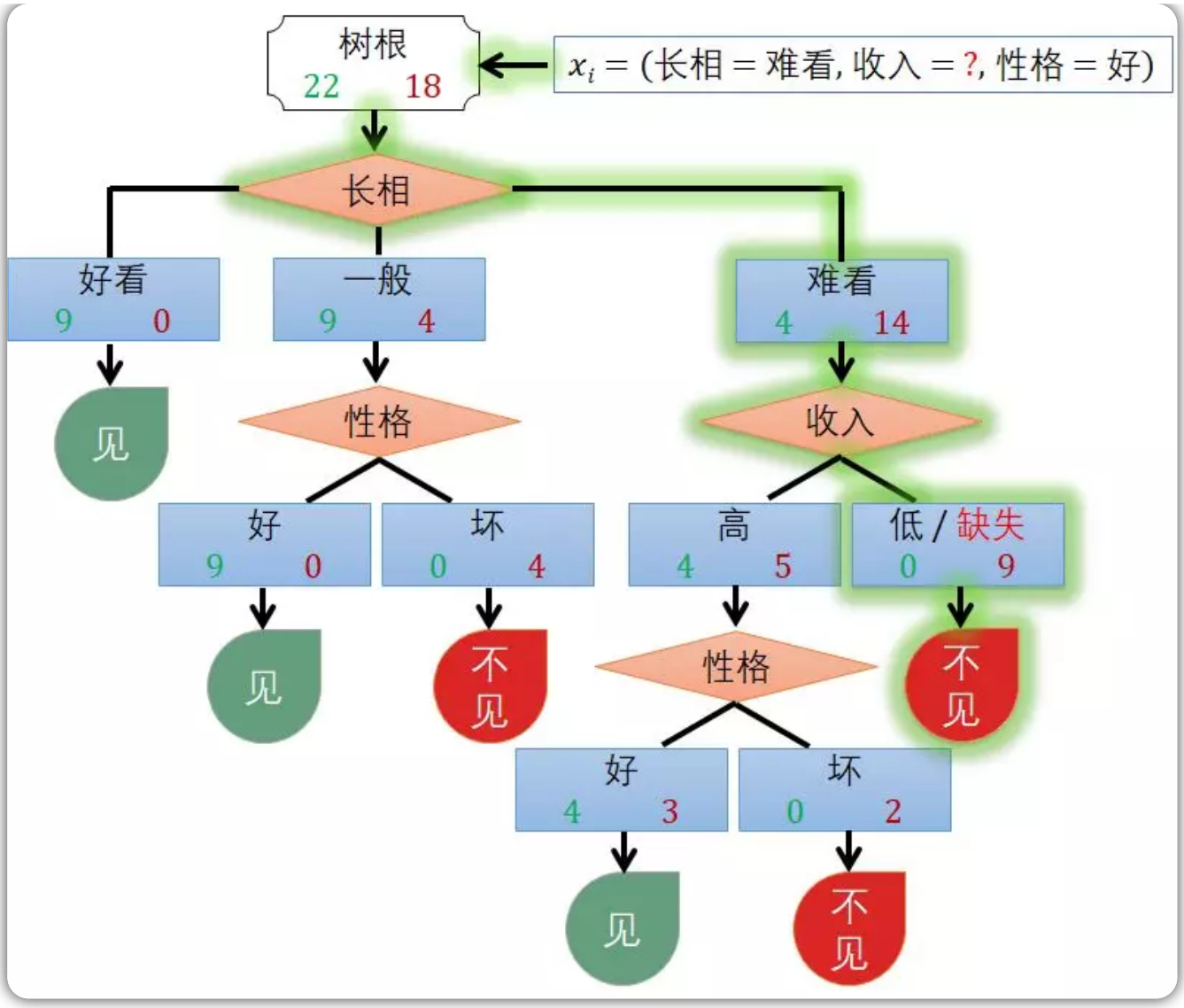
众数性格

均值收入

长相	性格	收入	见吗?
好看	好	50万	是
一般	坏	40万	否
一般	好	30万	是
难看	坏	5万	否
好看	坏	7万	否
一般	坏	10万	是
难看	坏	9万	否
好看	坏	20.4万	是
一般	坏	12万	是

归类法

归类的核心思想是把缺失 (unknown) 也当作是一种特征值。下图举例用决策树将“收入缺失”和“收入低”归纳成同一类。



这时缺失值是实实在在的一个类别了。

用 KNN 填充缺失值

这里介绍的填充缺失值的方法是用 k-近邻 (k-nearest neighbor, KNN) 来估算缺失值的，即在**每个特征**下，缺失值都是使用在训练集中找到 k 个最近邻居的平均值估算的。

代码如下 (引入 `sklearn.impute` 里面的 `KNNImputer`)：

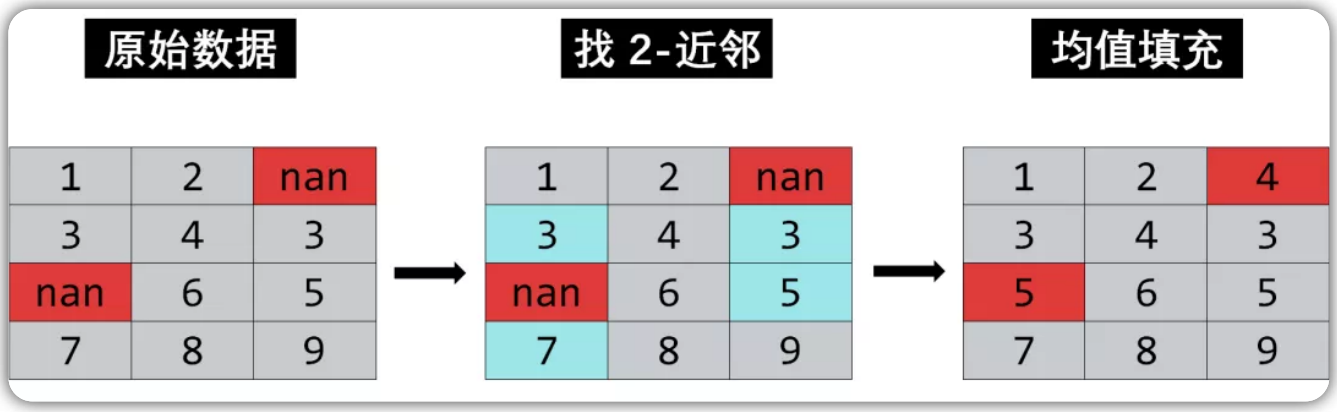
```
import numpy as np
from sklearn.impute import KNNImputer

X = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [7, 8, 9]]
imputer = KNNImputer(n_neighbors=2)
print(imputer.fit_transform(X))
```

```
[[1.  2.  4.]
 [3.  4.  3.]
 [5.  6.  5.]
 [7.  8.  9.]]
```

结果是合理的。X 的每一列代表一个特征，原始的 X 为

```
[[1.  2. nan]
 [3.  4. 3. ]
 [nan 6. 5. ]
 [7.  8. 9. ]]
```

在第一列中，离 nan 最近的 2 个邻居是 3 和 7，它们平均数是 **5**。在第四列中，离 nan 最近的 2 个邻居是 3 和 5，它们平均数是 **4**。总结图如下：



5 总结

回顾上面介绍的四个新填功能：

I. 一行画出 ROC-AUC 图，代码用

```
from sklearn.metrics import plot_roc_curve
```

II. 实现堆积法，代码用

```
from sklearn.ensemble import StackingClassifier
```

III. 为任何模型估计特征重要性，代码用

```
from sklearn.inspection import permutation_importance
```

IV. 用 k-近邻法来填充缺失值，代码用

```
from sklearn.impute import KNNImputer
```

Stay Tuned!