EDU

CSDN学院　IT实战派

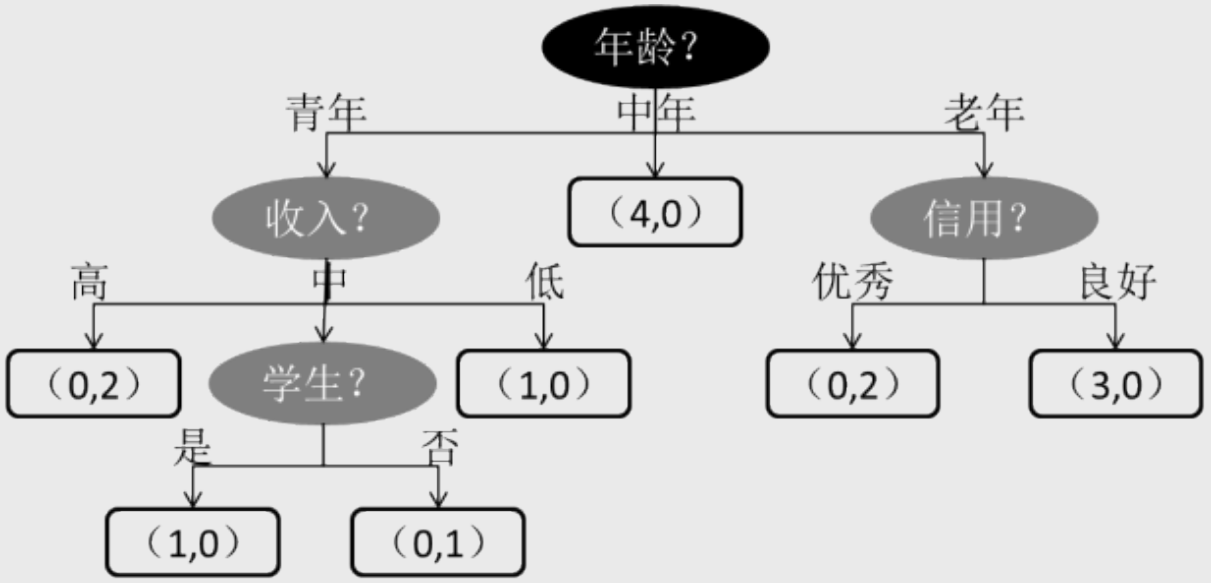
决策树与随机森林

讲师：刘顺祥

课程目标

1. 决策树节点字段的选择和阈值的选择
2. 随机森林的思想
3. 决策树与随机森林的实操

模型介绍



图中的决策树呈现自顶向下的生长过程，深色的椭圆表示树的根节点；浅色的椭圆表示树的中间节点；方框则表示树的叶节点。

对于所有的非叶节点来说，都是用来表示条件判断，而叶节点则存储最终的分类结果，例如中年分支下的叶节点（4,0）表示4位客户购买，0位客户不购买。

决策树节点字段的选择

信息熵

熵原本是物理学中的一个定义，后来香农将其引申到了信息论领域，用来表示信息量的大小。信息量越大（分类越不“纯净”），对应的熵值就越大，反之亦然。信息熵的计算公式如下：

$$H(p_1, p_2, \cdots p_k) = -\sum_{k=1}^{K} p_k log_2 p_k$$

在实际应用中，会将概率p_k的值用经验概率替换，所以经验信息熵可以表示为：

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} log_2 \frac{|C_k|}{|D|}$$

决策树节点字段的选择

信息熵

举例：以产品是否被购买为例，假设数据集一共包含14个样本，其中购买的用户有9个，没有购买的用户有5个，所以对于是否购买这个事件来说，它的经验信息熵为：

$$H(Buy) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.940$$

决策树节点字段的选择

信息熵

条件熵

$$H(D|A) = \sum_{i,k} P(A_i)\,H(D_k|A_i)$$

$$= -\sum_{i,k} P(A_i)P(D_k|A_i)log_2 P(D_k|A_i)\Big)$$

$$= -\sum_{i=1}^{n}\sum_{k=1}^{K} P(A_i)P(D_k|A_i)log_2 P(D_k|A_i)$$

$$= -\sum_{i=1}^{n} P(A_i)\sum_{k=1}^{K} P(D_k|A_i)log_2 P(D_k|A_i)$$

$$= -\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}log_2\frac{|D_{ik}|}{|D_i|}$$

其中，$P(A_i)$表示A事件的第i种值对应的概率；$H(D_k|A_i)$为已知A_i的情况下，D事件为k值的条件熵，其对应的计算公式为$P(D_k|A_i)log_2 P(D_k|A_i)$；$|D_i|$表示$A_i$的频数，$\frac{|D_i|}{|D|}$表示$A_i$在所有样本中的频率；$|D_{ik}|$表示$A_i$下$D$事件为$k$值的频数，$\frac{|D_{ik}|}{|D_i|}$表示所有$A_i$中，$D$事件为$k$值的频率。

决策树节点字段的选择

信息增益

信息增益

$$Gain_A(D) = H(D) - H(D|\mathrm{A})$$

对于已知的事件A来说，事件D的信息增益就是D的信息熵与A事件下D的条件熵之差，事件A对事件D的影响越大，条件熵$H(D|\mathrm{A})$就会越小（在事件A的影响下，事件D被划分得越“纯净”），体现在信息增益上就是差值越大，进而说明事件D的信息熵下降得越多。

所以，在根节点或中间节点的变量选择过程中，就是挑选出各自变量下因变量的信息增益最大的。

决策树节点字段的选择

信息增益率

决策树中的ID3算法使用信息增益指标实现根节点或中间节点的字段选择，但是该指标存在一个非常明显的缺点，即信息增益会偏向于取值较多的字段。

为了克服信息增益指标的缺点，提出了信息增益率的概念，它的思想很简单，就是在信息增益的基础上进行相应的惩罚。信息增益率的公式可以表示为：

$$Gain\_Ratio_A(D) = \frac{Gain_A(D)}{H_A}$$

其中，H_A为事件A的信息熵。事件A的取值越多，$Gain_A(D)$可能越大，但同时H_A也会越大，这样以商的形式就实现了$Gain_A(D)$的惩罚。

决策树节点字段的选择

基尼指数

决策树中的C4.5算法使用信息增益率指标实现根节点或中间节点的字段选择，但该算法与ID3算法一致，都只能针对离散型因变量进行分类，对于连续型的因变量就显得束手无策了。

为了能够让决策树预测连续型的因变量，Breiman等人在1984年提出了CART算法，该算法也称为分类回归树，它所使用的字段选择指标是基尼指数。

$$Gini(p_1, p_2, \cdots p_k) = \sum_{k=1}^{K} p_k(1-p_k) = \sum_{k=1}^{K} (p_k - {p_k}^2) = 1 - \sum_{k=1}^{K} {p_k}^2$$

$$Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2$$

基尼指数

条件基尼指数

$$Gini_A(D) = \sum_{i,k} P(A_i)\, Gini(D_k|A_i)$$

$$= \sum_{i=1}^{2} P(A_i)\left(1 - \sum_{k=1}^{K} (p_{ik})^2\right)$$

$$= \sum_{i=1}^{2} P\left(\frac{|D_i|}{|D|}\right)\left(1 - \sum_{k=1}^{K} \left(\frac{|D_{ik}|}{|D_i|}\right)^2\right)$$

其中，$P(A_i)$表示A变量在某个二元划分下第i组的概率，其对应的经验概率为$\frac{|D_i|}{|D|}$，即A变量中第i组的样本量与总样本量的商；$Gini(D_k|A_i)$表示在已知分组A_i的情况下，变量D取第k种值的条件基尼指数，其中$\frac{|D_{ik}|}{|D_i|}$表示分组A_i内变量D取第k种值的频率。

决策树节点字段的选择

基尼指数增益

与信息增益类似，还需要考虑自变量对因变量的影响程度，即因变量的基尼指数下降速度的快慢，下降得越快，自变量对因变量的影响就越强。下降速度的快慢可用下方式子衡量：

$$\triangle Gini(D) = Gini(D) - Gini_A(D)$$

决策树模型

函数说明

```
DecisionTreeClassifier(criterion='gini', splitter='best',
                       max_depth=None,min_samples_split=2,
                       min_samples_leaf=1,max_leaf_nodes=None,
                       class_weight=None)
```

criterion：用于指定选择节点字段的评价指标，对于分类决策树，默认为'gini'，表示采用基尼指数选择节点的最佳分割字段；对于回归决策树，默认为'mse'，表示使用均方误差选择节点的最佳分割字段
splitter：用于指定节点中的分割点选择方法，默认为'best'，表示从所有的分割点中选择最佳分割点；如果指定为'random'，则表示随机选择分割点
max_depth：用于指定决策树的最大深度，默认为None，表示树的生长过程中对深度不做任何限制
min_samples_split：用于指定根节点或中间节点能够继续分割的最小样本量， 默认为2
min_samples_leaf：用于指定叶节点的最小样本量，默认为1

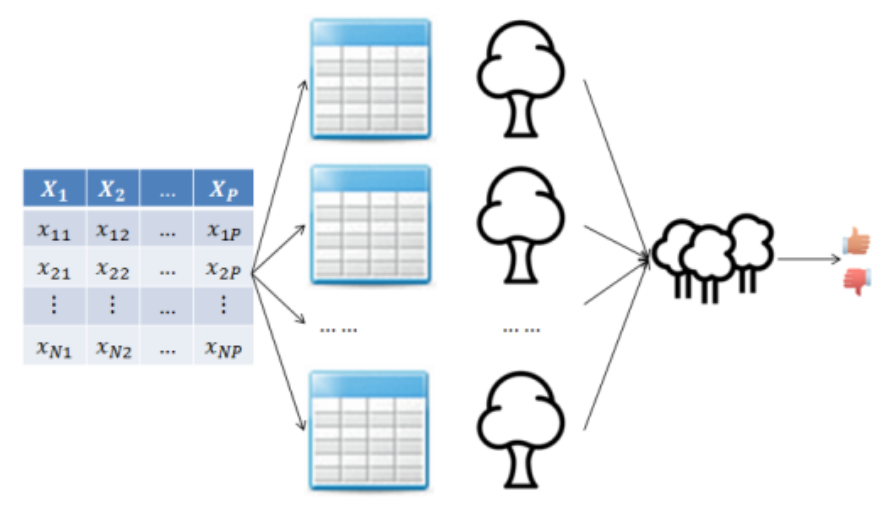
决策树模型

函数说明

```
DecisionTreeClassifier(criterion='gini', splitter='best',
                       max_depth=None,min_samples_split=2,
                       min_samples_leaf=1,max_leaf_nodes=None,
                       class_weight=None)
```

max_leaf_nodes：用于指定最大的叶节点个数，默认为None，表示对叶节点个数不做任何限制
class_weight：用于指定因变量中类别之间的权重，默认为None，表示每个类别的权重都相等；如果为balanced，则表示类别权重与原始样本中类别的比例成反比；还可以通过字典传递类别之间的权重差异，其形式为{class_label:weight}

随机森林的思想



利用Bootstrap抽样法，从原始数据集中生成k个数据集，并且每个数据集都含有N个观测和P个自变量。

针对每一个数据集，构造一棵CART决策树，在构建子树的过程中，并没有将所有自变量用作节点字段的选择，而是随机选择p个字段。

让每一棵决策树尽可能地充分生长，使得树中的每个节点尽可能“纯净”，即随机森林中的每一棵子树都不需要剪枝。

针对k棵CART树的随机森林，对分类问题利用投票法，将最高得票的类别用于最终的判断结果；对回归问题利用均值法，将其用作预测样本的最终结果。

随机森林模型

函数说明

```
RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
                       min_samples_split=2, min_samples_leaf=1,
                       max_leaf_nodes=None, bootstrap=True, class_weight=None)
```

n_estimators：用于指定随机森林所包含的决策树个数
criterion：用于指定每棵决策树节点的分割字段所使用的度量标准，用于分类的随机森林，默认的criterion值为'gini'；用于回归的随机森林，默认的criterion值为'mse'
max_depth：用于指定每棵决策树的最大深度，默认不限制树的生长深度
min_samples_split：用于指定每棵决策树根节点或中间节点能够继续分割的最小样本量， 默认为2

随机森林模型

函数说明

```
RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
                       min_samples_split=2, min_samples_leaf=1,
                       max_leaf_nodes=None, bootstrap=True, class_weight=None)
```

min_samples_leaf：用于指定每棵决策树叶节点的最小样本量，默认为1
max_leaf_nodes：用于指定每棵决策树最大的叶节点个数，默认为None，表示对叶节点个数不做任何限制
bootstrap：bool类型参数，是否对原始数据集进行bootstrap抽样，用于子树的构建，默认为True
class_weight：用于指定因变量中类别之间的权重，默认为None，表示每个类别的权重都相等

代码演示

```
# 读入数据
Titanic = pd.read_csv(r'C:\Users\Administrator\Desktop\Titanic.csv')
# 删除无意义的变量，并检查剩余变量是否含有缺失值
Titanic.drop(['PassengerId','Name','Ticket','Cabin'], axis = 1, inplace = True)
Titanic.isnull().sum(axis = 0)

# 对Sex分组，用各组乘客的平均年龄填充各组中的缺失年龄
fillna_Titanic = []
for i in Titanic.Sex.unique():
    update = Titanic.loc[Titanic.Sex == i,].fillna(value = {'Age': Titanic.Age[Titanic.Sex == i].mean()}, inplace
= False)
    fillna_Titanic.append(update)
Titanic = pd.concat(fillna_Titanic)
# 使用Embarked变量的众数填充缺失值
Titanic.fillna(value = {'Embarked':Titanic.Embarked.mode()[0]}, inplace=True)
```

代码演示

```
# 将数值型的Pclass转换为类别型，否则无法对其哑变量处理
Titanic.Pclass = Titanic.Pclass.astype('category')
# 哑变量处理
dummy = pd.get_dummies(Titanic[['Sex','Embarked','Pclass']])
# 水平合并Titanic数据集和哑变量的数据集
Titanic = pd.concat([Titanic,dummy], axis = 1)
# 删除原始的Sex、Embarked和Pclass变量
Titanic.drop(['Sex','Embarked','Pclass'], inplace=True, axis = 1)

# 取出所有自变量名称
predictors = Titanic.columns[1:]
# 将数据集拆分为训练集和测试集，且测试集的比例为25%
X_train, X_test, y_train, y_test = model_selection.train_test_split(Titanic[predictors], Titanic.Survived,
test_size = 0.25, random_state = 1234)
```

代码演示

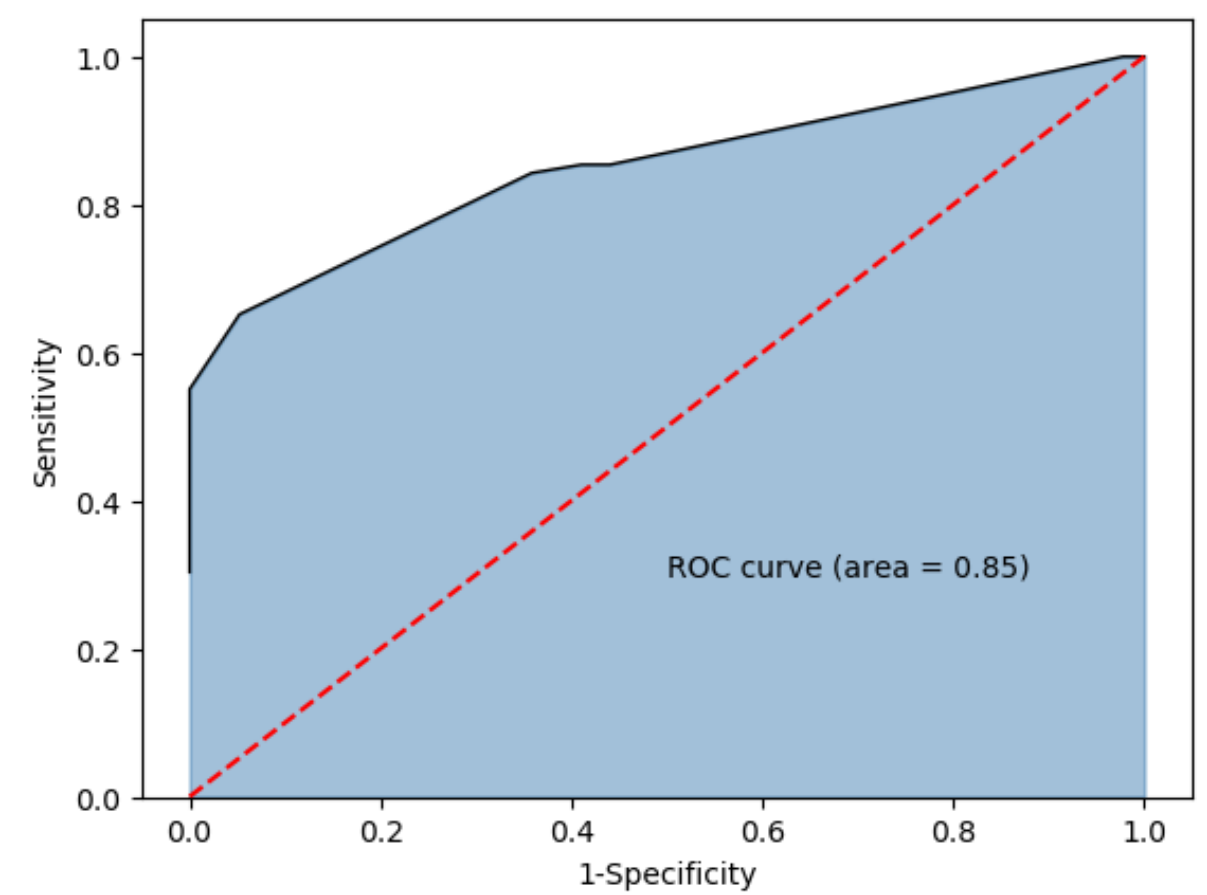
```
# 预设各参数的不同选项值
max_depth = [2,3,4,5,6]
min_samples_split = [2,4,6,8]
min_samples_leaf = [2,4,8,10,12]
# 将各参数值以字典形式组织起来
parameters = {'max_depth':max_depth, 'min_samples_split':min_samples_split,
              'min_samples_leaf':min_samples_leaf}
# 网格搜索法，测试不同的参数值
grid_dtcateg = GridSearchCV(estimator = tree.DecisionTreeClassifier(), param_grid = parameters, cv=10)
# 模型拟合
grid_dtcateg.fit(X_train, y_train)
# 返回最佳组合的参数值
grid_dtcateg.best_params

out:
{'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 2}
```

代码演示

```
# 构建分类决策树
CART_Class = tree.DecisionTreeClassifier(max_depth=3, min_samples_leaf=4, min_samples_split=2)
# 模型拟合
decision_tree = CART_Class.fit(X_train, y_train)
# 模型在测试集上的预测
pred = CART_Class.predict(X_test)
# 模型的准确率
print('模型在测试集的预测准确率：\n',metrics.accuracy_score(y_test, pred))
print('模型在训练集的预测准确率：\n',
      metrics.accuracy_score(y_train, CART_Class.predict(X_train)))
```

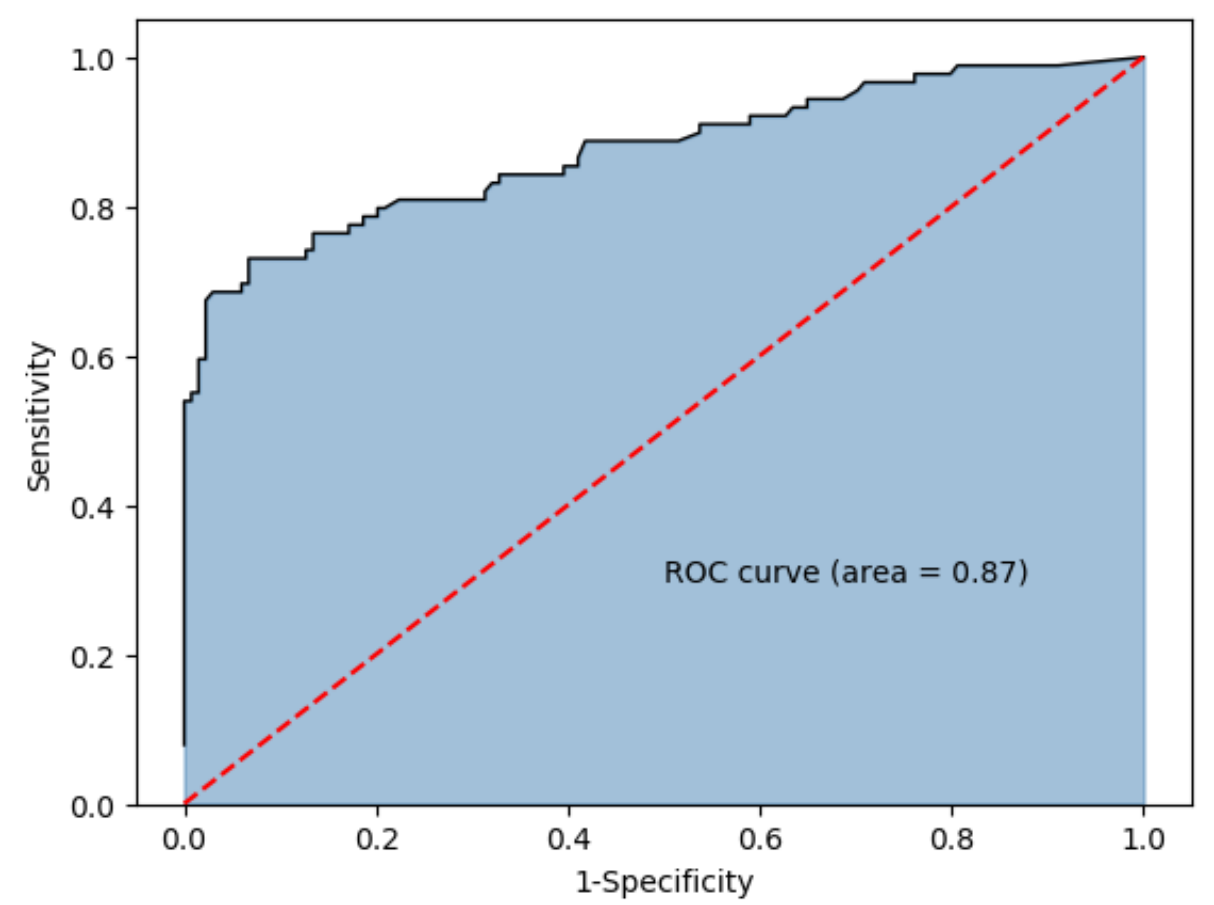
out:

```
模型在测试集的预测准确率：
 0.829596412556
```



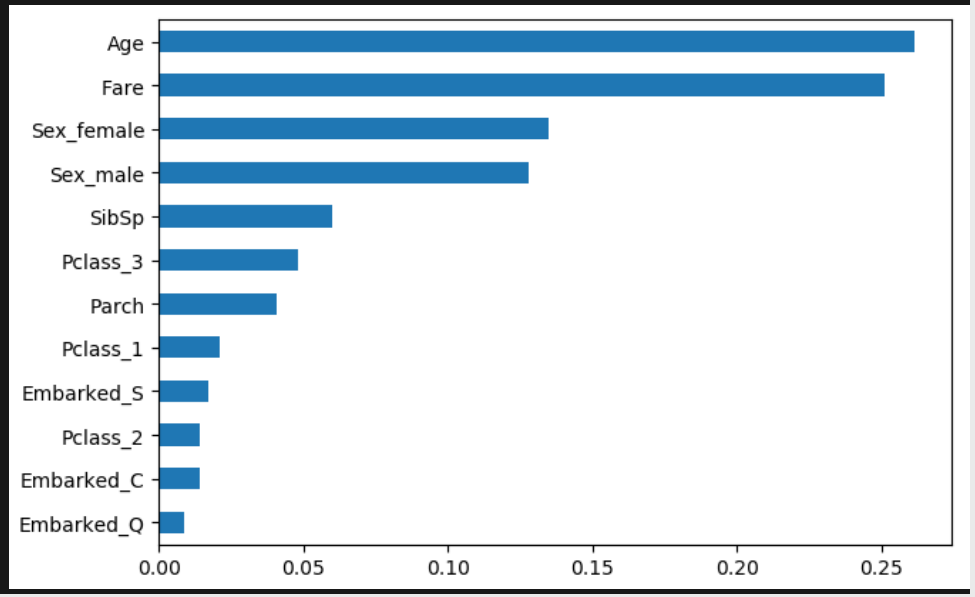
代码演示

```
# 构建随机森林
RF_class = ensemble.RandomForestClassifier(n_estimators=200, random_state=1234)
# 随机森林的拟合
RF_class.fit(X_train, y_train)
# 模型在测试集上的预测
RFclass_pred = RF_class.predict(X_test)
# 模型的准确率
print('模型在测试集的预测准确率：\n',metrics.accuracy_score(y_test, RFclass_pred))

out:
模型在测试集的预测准确率：
 0.85201793722
```



代码演示

```
# 变量的重要性程度值
importance = RF_class.feature_importances_
# 构建含序列用于绘图
Impt_Series = pd.Series(importance, index = X_train.columns)
# 对序列排序绘图
Impt_Series.sort_values(ascending = True).plot('barh')
# 显示图形
plt.show()
```



EDU

CSDN学院　IT实战派