决策树回归

决策树

基本算法原理

核心思想：相似的输入必会产生相似的输出。例如预测某人薪资：

年龄：1-青年，2-中年，3-老年
学历：1-本科，2-硕士，3-博士
经历：1-出道，2-一般，3-老手，4-骨灰
性别：1-男性，2-女性

年龄	学历	经历	性别	==>	薪资
1	1	1	1	==>	6000（低）
2	1	3	1	==>	10000（中）
3	3	4	1	==>	50000（高）
...	...	...	...	==>	...
1	3	2	2	==>	?

为了提高搜索效率，使用树形数据结构处理样本数据：

$$\text{年龄}=1\begin{cases}\text{学历}1\\ \text{学历}2\\ \text{学历}3\end{cases}\qquad \text{年龄}=2\begin{cases}\text{学历}1\\ \text{学历}2\\ \text{学历}3\end{cases}\qquad \text{年龄}=3\begin{cases}\text{学历}1\\ \text{学历}2\\ \text{学历}3\end{cases}$$

首先从训练样本矩阵中选择一个特征进行子表划分，使每个子表中该特征的值全部相同，然后再在每个子表中选择下一个特征按照同样的规则继续划分更小的子表，不断重复直到所有的特征全部使用完为止，此时便得到叶级子表，其中所有样本的特征值全部相同。对于待预测样本，根据其每一个特征的值，选择对应的子表，逐一匹配，直到找到与之完全匹配的叶级子表，用该子表中样本的输出，通过平均(回归)或者投票(分类)为待预测样本提供输出。

首先选择哪一个特征进行子表划分决定了决策树的性能。这么多特征，使用哪个特征先进行子表划分?

sklearn提供的决策树底层为cart树（Classification and Regression Tree），cart回归树在解决回归问题时的步骤如下：

1. 原始数据集S，此时树的深度depth=0；
2. 针对集合S，遍历每一个特征的每一个value(遍历数据中的所有离散值 (12个）)

   用该value将原数据集S分裂成2个集合：左集合left(<=value的样本)、右集合right(>value的样本)，

分别计算这2个集合的mse(均方误差)，找到使（left_mse+right_mse）最小的那个value，记录下此时的特征名称和value，这个就是最佳分割特征以及最佳分割值；

3.找到最佳分割特征以及最佳分割value之后，用该value将集合S分裂成2个集合，depth+=1；

4.针对集合left、right分别重复步骤2,3，直到达到终止条件。

决策树底层结构 为二叉树

终止条件有如下几种：
1、特征已经用完了：没有可供使用的特征再进行分裂了，则树停止分裂；
2、子节点中没有样本了：此时该结点已经没有样本可供划分，该结点停止分裂；
3、树达到了人为预先设定的最大深度：depth >= max_depth，树停止分裂。
4、节点的样本数量达到了人为设定的阈值：样本数量 < min_samples_split ，则该节点停止分裂；

决策树回归器模型相关API：

```
import sklearn.tree as st

# 创建决策树回归器模型 决策树的最大深度为4
model = st.DecisionTreeRegressor(max_depth=4)
# 训练模型
# train_x： 二维数组样本数据
# train_y： 训练集中对应每行样本的结果
model.fit(train_x, train_y)
# 测试模型
pred_test_y = model.predict(test_x)
```

案例：预测波士顿地区房屋价格。

1. 读取数据，打断原始数据集。 划分训练集和测试集。

```
import sklearn.datasets as sd
import sklearn.utils as su
# 加载波士顿地区房价数据集
boston = sd.load_boston()
print(boston.feature_names)
# |CRIM|ZN|INDUS|CHAS|NOX|RM|AGE|DIS|RAD|TAX|PTRATIO|B|LSTAT|
# 犯罪率|住宅用地比例|商业用地比例|是否靠河|空气质量|房间数|年限|距中心区距离|路网密度|房产税|师生比|黑人比例|低地位人口比例|
# 打乱原始数据集的输入和输出
x, y = su.shuffle(boston.data, boston.target, random_state=7)
# 划分训练集和测试集
train_size = int(len(x) * 0.8)
train_x, test_x, train_y, test_y = \
    x[:train_size], x[train_size:], \
    y[:train_size], y[train_size:]
```

1. 创建决策树回归器模型，使用训练集训练模型。使用测试集测试模型。

```
import sklearn.tree as st
import sklearn.metrics as sm

# 创建决策树回归模型
model = st.DecisionTreeRegressor(max_depth=4)
# 训练模型
model.fit(train_x, train_y)
# 测试模型
pred_test_y = model.predict(test_x)
print(sm.r2_score(test_y, pred_test_y))
```

集合算法(集成学习)

单个模型得到的预测结果总是片面的，根据多个不同模型给出的预测结果，利用平均(回归)或者投票(分类)的方法，得出最终预测结果。

基于决策树的集合算法，就是按照某种规则，构建多棵彼此不同的决策树模型，分别给出针对未知样本的预测结果，最后通过平均或投票得到相对综合的结论。常用的集合模型包括Boosting类模型（AdaBoost、GBDT）与Bagging（自助聚合、随机森林）类模型。

AdaBoost模型（正向激励）

首先为样本矩阵中的样本随机分配初始权重，由此构建一棵带有权重的决策树，在由该决策树提供预测输出时，通过加权平均或者加权投票的方式产生预测值。

自适应增强决策树

```
已经构建好一个决策树 通过1322 找到所有的女博士 一个4个 6000 8000 9000 10000
由于正向激励，对每个样本都分配了初始权重 权重为:1 1 1 3 预测： 加权均值
```

将训练样本代入模型，预测其输出，对那些预测值与实际值不同的样本，提高其权重，由此形成第二棵决策树。重复以上过程，构建出不同权重的若干棵决策树。

```
实际值：10000 但是你预测的为6000 构建第二个决策树，提高10000样本的权重
```

不断的提高真实样本的权重值,让预测结果越来越接近真实值

正向激励相关API：

```
import sklearn.tree as st
import sklearn.ensemble as se
# model: 决策树模型（一颗）
```

```
model = st.DecisionTreeRegressor(max_depth=4)
# 自适应增强决策树回归模型
#
model = se.AdaBoostRegressor(model, n_estimators=400, random_state=7)

正向激励 的基础模型： 决策树
n_estimators：构建400棵不同权重的决策树，训练模型

# 训练模型
model.fit(train_x, train_y)
# 测试模型
pred_test_y = model.predict(test_x)
```

案例：基于正向激励训练预测波士顿地区房屋价格的模型。

```
# 创建基于决策树的正向激励回归器模型
model = se.AdaBoostRegressor(
    st.DecisionTreeRegressor(max_depth=4), n_estimators=400, random_state=7)
# 训练模型
model.fit(train_x, train_y)
# 测试模型
pred_test_y = model.predict(test_x)
print(sm.r2_score(test_y, pred_test_y))
```

特征重要性(重要程度)

作为决策树模型训练过程的副产品，根据划分子表时选择特征的顺序标志了该特征的重要程度，此即为该特征重要性指标。训练得到的模型对象提供了属性：feature_importances_来存储每个特征的重要性。

获取样本矩阵特征重要性属性：

```
model.fit(train_x, train_y)
fi = model.feature_importances_
```

案例：获取普通决策树与正向激励决策树训练的两个模型的特征重要性值，按照从大到小顺序输出绘图。

```
import matplotlib.pyplot as mp

model = st.DecisionTreeRegressor(max_depth=4)
model.fit(train_x, train_y)
# 决策树回归器给出的特征重要性
fi_dt = model.feature_importances_
model = se.AdaBoostRegressor(
    st.DecisionTreeRegressor(max_depth=4), n_estimators=400, random_state=7)
model.fit(train_x, train_y)
# 基于决策树的正向激励回归器给出的特征重要性
fi_ab = model.feature_importances_


mp.figure('Feature Importance', facecolor='lightgray')
```

```
mp.subplot(211)
mp.title('Decision Tree', fontsize=16)
mp.ylabel('Importance', fontsize=12)
mp.tick_params(labelsize=10)
mp.grid(axis='y', linestyle=':')
sorted_indices = fi_dt.argsort()[::-1]
pos = np.arange(sorted_indices.size)
mp.bar(pos, fi_dt[sorted_indices], facecolor='deepskyblue', edgecolor='steelblue')
mp.xticks(pos, feature_names[sorted_indices], rotation=30)
mp.subplot(212)
mp.title('AdaBoost Decision Tree', fontsize=16)
mp.ylabel('Importance', fontsize=12)
mp.tick_params(labelsize=10)
mp.grid(axis='y', linestyle=':')
sorted_indices = fi_ab.argsort()[::-1]
pos = np.arange(sorted_indices.size)
mp.bar(pos, fi_ab[sorted_indices], facecolor='lightcoral', edgecolor='indianred')
mp.xticks(pos, feature_names[sorted_indices], rotation=30)
mp.tight_layout()
mp.show()
```

GBDT

GBDT（Gradient Boosting Decision Tree 梯度提升树）通过多轮迭代，每轮迭代产生一个弱分类器，每个分类器在上一轮分类器的残差**（残差在数理统计中是指实际观察值与估计值（拟合值）之间的差）**基础上进行训练。基于预测结果的残差设计损失函数。GBDT训练的过程即是求该损失函数最小值的过程。

拟合残差 y-y' 不断的接近于0

案例

案例：预测年龄

样本	消费金额	上网时长	年龄
A	1000	1	14
B	800	1.2	16
C	1200	0.9	24
D	1400	1.5	26

GBDT原理

原理1：

样本	消费金额	上网时长	年龄
A	1000	1	14
B	800	1.2	16
C	1200	0.9	24
D	1400	1.5	26

训练第一颗决策树：

$$\text{总样本}\begin{cases}(\text{金额} <= 1000) \Rightarrow \begin{bmatrix} A & 1000 & 1 & 14 \\ B & 800 & 1.2 & 16 \end{bmatrix} \\ (\text{金额} > 1000) \Rightarrow \begin{bmatrix} C & 1200 & 1 & 24 \\ D & 1400 & 1.5 & 26 \end{bmatrix}\end{cases}$$

计算每个样本的真实结果与预测结果之差（残差）：

样本	消费金额	上网时长	残差
A	1000	1	-1
B	800	1.2	1
C	1200	0.9	-1
D	1400	1.5	1

原理2：

样本	消费金额	上网时长	残差
A	1000	1	-1
B	800	1.2	1
C	1200	0.9	-1
D	1400	1.5	1

基于残差训练第二颗决策树：

$$\text{总样本}\begin{cases}(\text{时长} <= 1) \Rightarrow \begin{bmatrix} A & 1000 & 1 & -1 \\ C & 1200 & 1 & -1 \end{bmatrix} \\ (\text{时长} > 1) \Rightarrow \begin{bmatrix} B & 800 & 1.2 & 1 \\ D & 1400 & 1.5 & 1 \end{bmatrix}\end{cases}$$

原理3：

预测过程：预测以下样本的年龄：

样本	消费金额	上网时长	年龄
E	1200	1.6	?

$$\text{总样本}\begin{cases}(\text{金额} <= 1000) \Rightarrow \begin{bmatrix} A & 1000 & 1 & 14 \\ B & 800 & 1.2 & 16 \end{bmatrix} \\ (\text{金额} > 1000) \Rightarrow \begin{bmatrix} C & 1200 & 1 & 24 \\ D & 1400 & 1.5 & 26 \end{bmatrix}\end{cases} \quad \text{总样本}\begin{cases}(\text{时长} <= 1) \Rightarrow \begin{bmatrix} A & 1000 & 1 & -1 \\ C & 1200 & 1 & -1 \end{bmatrix} \\ (\text{时长} > 1) \Rightarrow \begin{bmatrix} B & 800 & 1.2 & 1 \\ D & 1400 & 1.5 & 1 \end{bmatrix}\end{cases}$$

将每颗决策树的预测结果相加：25+1=26

```
import sklearn.tree as st
import sklearn.ensemble as se
# 自适应增强决策树回归模型
# n_estimators：构建400棵不同权重的决策树，训练模型
model = se.GridientBoostingRegressor(
    max_depth=10, n_estimators=1000, min_samples_split=2)
# 训练模型
model.fit(train_x, train_y)
# 测试模型
pred_test_y = model.predict(test_x)
```

boosting : Adaboost GBDT

自助聚合

每次从总样本矩阵中以有放回抽样的方式随机抽取部分样本构建决策树，这样形成多棵包含不同训练样本的决策树，以削弱某些强势样本对模型预测结果的影响，提高模型的泛化特性。

随机森林

在自助聚合的基础上，每次构建决策树模型时，不仅随机选择部分样本，而且还随机选择部分特征，这样的集合算法，不仅规避了强势样本对预测结果的影响，而且也削弱了强势特征的影响，使模型的预测能力更加泛化。

随机森林相关API：

```
import sklearn.ensemble as se
# 随机森林回归模型 （属于集合算法的一种）
# max_depth：决策树最大深度10
# n_estimators：构建1000棵决策树，训练模型
# min_samples_split: 子表中最小样本数 若小于这个数字，则不再继续向下拆分
model = se.RandomForestRegressor(
    max_depth=10, n_estimators=1000, min_samples_split=2)
```

案例：分析共享单车的需求，从而判断如何进行共享单车的投放。

```
1。加载并整理数据集
2.特征分析
3.打乱数据集，划分训练集，测试集
```

```
import numpy as np
import sklearn.utils as su
import sklearn.ensemble as se
import sklearn.metrics as sm
import matplotlib.pyplot as mp

data = np.loadtxt('../data/bike_day.csv', unpack=False, dtype='U20', delimiter=',')
day_headers = data[0, 2:13]
x = np.array(data[1:, 2:13], dtype=float)
y = np.array(data[1:, -1], dtype=float)

x, y = su.shuffle(x, y, random_state=7)
print(x.shape, y.shape)
train_size = int(len(x) * 0.9)
train_x, test_x, train_y, test_y = \
    x[:train_size], x[train_size:], y[:train_size], y[train_size:]
# 随机森林回归器
model = se.RandomForestRegressor( max_depth=10, n_estimators=1000, min_samples_split=2)
model.fit(train_x, train_y)
# 基于“天”数据集的特征重要性
fi_dy = model.feature_importances_
pred_test_y = model.predict(test_x)
print(sm.r2_score(test_y, pred_test_y))

data = np.loadtxt('../data/bike_hour.csv', unpack=False, dtype='U20', delimiter=',')
hour_headers = data[0, 2:13]
x = np.array(data[1:, 2:13], dtype=float)
y = np.array(data[1:, -1], dtype=float)
x, y = su.shuffle(x, y, random_state=7)
train_size = int(len(x) * 0.9)
train_x, test_x, train_y, test_y = \
    x[:train_size], x[train_size:], \
    y[:train_size], y[train_size:]
# 随机森林回归器
model = se.RandomForestRegressor(
```

```
    max_depth=10, n_estimators=1000,
    min_samples_split=2)
model.fit(train_x, train_y)
# 基于“小时”数据集的特征重要性
fi_hr = model.feature_importances_
pred_test_y = model.predict(test_x)
print(sm.r2_score(test_y, pred_test_y))
```

画图显示两组样本数据的特征重要性：

```
mp.figure('Bike', facecolor='lightgray')
mp.subplot(211)
mp.title('Day', fontsize=16)
mp.ylabel('Importance', fontsize=12)
mp.tick_params(labelsize=10)
mp.grid(axis='y', linestyle=':')
sorted_indices = fi_dy.argsort()[::-1]
pos = np.arange(sorted_indices.size)
mp.bar(pos, fi_dy[sorted_indices], facecolor='deepskyblue', edgecolor='steelblue')
mp.xticks(pos, day_headers[sorted_indices], rotation=30)

mp.subplot(212)
mp.title('Hour', fontsize=16)
mp.ylabel('Importance', fontsize=12)
mp.tick_params(labelsize=10)
mp.grid(axis='y', linestyle=':')
sorted_indices = fi_hr.argsort()[::-1]
pos = np.arange(sorted_indices.size)
mp.bar(pos, fi_hr[sorted_indices], facecolor='lightcoral', edgecolor='indianred')
mp.xticks(pos, hour_headers[sorted_indices], rotation=30)
mp.tight_layout()
mp.show()
```

-