机器学习DAY04

集合算法

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。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。

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
# n_estimators：构建400棵不同权重的决策树，训练模型
model = se.AdaBoostRegressor(model, 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)
```

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

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。

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

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

基于残差训练第二颗决策树：

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

再次计算每个样本残差为0，再无优化空间，迭代结束。

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	?

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.GradientBoostingRegressor(
        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)
```

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。

```
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()
```

分类模型

什么问题属于分类问题？

sklearn.datasets.load_iris()　鸢尾花数据集

逻辑回归

逻辑回归分类模型是一种基于回归思想实现分类业务的分类模型。

逻辑回归做二元分类时的核心思想为：

针对输出为{0, 1}的已知训练样本训练一个回归模型，使得训练样本的预测输出限制在(0, 1)的数值区间。该使原类别为0的样本的输出更接近于0，原类别为1的样本的输出更接近于1。这样就可以使用相同的回归模型来完成分类预测。

逻辑回归原理：

逻辑回归目标函数：

$$\text{逻辑函数}(sigmoid): y = \frac{1}{1+e^{-z}}; \quad z = w^T x + b$$

该逻辑函数值域被限制在(0, 1)区间，当x>0，y>0.5；当x<0, y<0.5。可以把训练样本数据通过线性预测模型 z 代入逻辑函数，找到一组最优秀的模型参数使得原本属于1类别的样本输出趋近于1；原本属于0类别的样本输出趋近于0。即将预测函数的输出看做被划分为1类的概率，择概率大的类别作为预测结果。

逻辑回归相关API：

```
import sklearn.linear_model as lm
# 构建逻辑回归器
# solver：逻辑函数中指数的函数关系（liblinear为线型函数关系）
# C：参数代表正则强度，为了防止过拟合。正则越大拟合效果越小。
model = lm.LogisticRegression(solver='liblinear', C=正则强度)
model.fit(训练输入集，训练输出集)
result = model.predict(带预测输入集)
```

案例：基于逻辑回归器绘制网格化坐标颜色矩阵。

```
import numpy as np
import sklearn.linear_model as lm
import matplotlib.pyplot as mp
x = np.array([
    [3, 1],
    [2, 5],
    [1, 8],
    [6, 4],
    [5, 2],
    [3, 5],
    [4, 7],
    [4, -1]])
y = np.array([0, 1, 1, 0, 0, 1, 1, 0])`
# 逻辑分类器
model = lm.LogisticRegression(solver='liblinear', C=1)
model.fit(x, y)
l, r = x[:, 0].min() - 1, x[:, 0].max() + 1
b, t = x[:, 1].min() - 1, x[:, 1].max() + 1
n = 500
grid_x, grid_y = np.meshgrid(np.linspace(l, r, n), np.linspace(b, t, n))
samples = np.column_stack((grid_x.ravel(), grid_y.ravel()))

grid_z = model.predict(samples)
grid_z = grid_z.reshape(grid_x.shape)
```

```
mp.figure('Logistic Classification', facecolor='lightgray')
mp.title('Logistic Classification', fontsize=20)
mp.xlabel('x', fontsize=14)
mp.ylabel('y', fontsize=14)
mp.tick_params(labelsize=10)
mp.pcolormesh(grid_x, grid_y, grid_z, cmap='gray')
mp.scatter(x[:, 0], x[:, 1], c=y, cmap='brg', s=80)
mp.show()
```

多元分类

通过多个二元分类器解决多元分类问题。

特征1	特征2	==>	所属类别
4	7	==>	A
3.5	8	==>	A
1.2	1.9	==>	B
5.4	2.2	==>	C

若拿到一组新的样本，可以基于二元逻辑分类训练出一个模型判断属于A类别的概率。再使用同样的方法训练出两个模型分别判断属于B、C类型的概率，最终选择概率最高的类别作为新样本的分类结果。

案例：基于逻辑分类模型的多元分类。

```
import numpy as np
import sklearn.linear_model as lm
import matplotlib.pyplot as mp
x = np.array([
    [4, 7],
    [3.5, 8],
    [3.1, 6.2],
    [0.5, 1],
    [1, 2],
    [1.2, 1.9],
    [6, 2],
    [5.7, 1.5],
    [5.4, 2.2]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2])
# 逻辑分类器
model = lm.LogisticRegression(solver='liblinear', C=1000)
model.fit(x, y)
l, r = x[:, 0].min() - 1, x[:, 0].max() + 1
b, t = x[:, 1].min() - 1, x[:, 1].max() + 1
n = 500
grid_x, grid_y = np.meshgrid(np.linspace(l, r, n), np.linspace(b, t, n))
samples = np.column_stack((grid_x.ravel(), grid_y.ravel()))
grid_z = model.predict(samples)
print(grid_z)
grid_z = grid_z.reshape(grid_x.shape)

mp.figure('Logistic Classification', facecolor='lightgray')
mp.title('Logistic Classification', fontsize=20)
mp.xlabel('x', fontsize=14)
```

```
mp.ylabel('y', fontsize=14)
mp.tick_params(labelsize=10)
mp.pcolormesh(grid_x, grid_y, grid_z, cmap='gray')
mp.scatter(x[:, 0], x[:, 1], c=y, cmap='brg', s=80)
mp.show()
```

数据集划分

对于分类问题训练集和测试集的划分不应该用整个样本空间的特定百分比作为训练数据，而应该在其每一个类别的样本中抽取特定百分比作为训练数据。sklearn模块提供了数据集划分相关方法，可以方便的划分训练集与测试集数据，使用不同数据集训练或测试模型，达到提高分类可信度。

数据集划分相关API：

```
import sklearn.model_selection as ms

训练输入, 测试输入, 训练输出, 测试输出 = \
    ms.train_test_split(
        输入集, 输出集, test_size=测试集占比, random_state=随机种子)

```

案例：

```
import numpy as np
import sklearn.model_selection as ms
import sklearn.naive_bayes as nb
import matplotlib.pyplot as mp
data = np.loadtxt('../data/multiple1.txt', unpack=False, dtype='U20',
delimiter=',')
print(data.shape)
x = np.array(data[:, :-1], dtype=float)
y = np.array(data[:, -1], dtype=float)
# 划分训练集和测试集
train_x, test_x, train_y, test_y = \
    ms.train_test_split( x, y, test_size=0.25, random_state=7)
# 朴素贝叶斯分类器
model = nb.GaussianNB()
# 用训练集训练模型
model.fit(train_x, train_y)

l, r = x[:, 0].min() - 1, x[:, 0].max() + 1
b, t = x[:, 1].min() - 1, x[:, 1].max() + 1
n = 500
grid_x, grid_y = np.meshgrid(np.linspace(l, r, n), np.linspace(b, t, n))
samples = np.column_stack((grid_x.ravel(), grid_y.ravel()))
grid_z = model.predict(samples)
grid_z = grid_z.reshape(grid_x.shape)

pred_test_y = model.predict(test_x)
# 计算并打印预测输出的精确度
print((test_y == pred_test_y).sum() / pred_test_y.size)

mp.figure('Naive Bayes Classification', facecolor='lightgray')
mp.title('Naive Bayes Classification', fontsize=20)
mp.xlabel('x', fontsize=14)
mp.ylabel('y', fontsize=14)
```

```
mp.tick_params(labelsize=10)
mp.pcolormesh(grid_x, grid_y, grid_z, cmap='gray')
mp.scatter(test_x[:,0], test_x[:,1], c=test_y, cmap='brg', s=80)
mp.show()
```

交叉验证

由于数据集的划分有不确定性，若随机划分的样本正好处于某类特殊样本，则得到的训练模型所预测的结果的可信度将受到质疑。所以需要进行多次交叉验证，把样本空间中的所有样本均分成n份，使用不同的训练集训练模型，对不同的测试集进行测试时输出指标得分。sklearn提供了交叉验证相关API：

```
import sklearn.model_selection as ms
指标值数组 = \
    ms.cross_val_score(模型, 输入集, 输出集, cv=折叠数, scoring=指标名)
```

案例：使用交叉验证，输出分类器的精确度：

```
# 划分训练集和测试集
train_x, test_x, train_y, test_y = \
    ms.train_test_split(
        x, y, test_size=0.25, random_state=7)
# 朴素贝叶斯分类器
model = nb.GaussianNB()
# 交叉验证
# 精确度
ac = ms.cross_val_score( model, train_x, train_y, cv=5, scoring='accuracy')
print(ac.mean())
#用训练集训练模型
model.fit(train_x, train_y)
```

交叉验证指标

1. 精确度(accuracy)：分类正确的样本数/总样本数
2. 查准率(precision_weighted)：针对每一个类别，预测正确的样本数比上预测出来的样本数
3. 召回率(recall_weighted)：针对每一个类别，预测正确的样本数比上实际存在的样本数
4. f1得分(f1_weighted)：

   2x查准率x召回率/(查准率+召回率)

在交叉验证过程中，针对每一次交叉验证，计算所有类别的查准率、召回率或者f1得分，然后取各类别相应指标值的平均数，作为这一次交叉验证的评估指标，然后再将所有交叉验证的评估指标以数组的形式返回调用者。

```
# 交叉验证
# 精确度
ac = ms.cross_val_score( model, train_x, train_y, cv=5, scoring='accuracy')
print(ac.mean())
# 查准率
pw = ms.cross_val_score( model, train_x, train_y, cv=5,
scoring='precision_weighted')
print(pw.mean())
# 召回率
rw = ms.cross_val_score( model, train_x, train_y, cv=5,
scoring='recall_weighted')
print(rw.mean())
# f1得分
fw = ms.cross_val_score( model, train_x, train_y, cv=5,
scoring='f1_weighted')
print(fw.mean())
```

混淆矩阵

每一行和每一列分别对应样本输出中的每一个类别，行表示实际类别，列表示预测类别。

	A类别	B类别	C类别
A类别	3	1	1
B类别	0	4	2
C类别	0	0	7

上述矩阵即为不理想的混淆矩阵。理想的混淆矩阵如下：

	A类别	B类别	C类别
A类别	5	0	0
B类别	0	6	0
C类别	0	0	7

查准率 = 主对角线上的值 / 该值所在列的和

召回率 = 主对角线上的值 / 该值所在行的和

获取模型分类结果的混淆矩阵的相关API：

```
import sklearn.metrics as sm
混淆矩阵 = sm.confusion_matrix(实际输出, 预测输出)
```

案例：输出分类结果的混淆矩阵。

```
#输出混淆矩阵并绘制混淆矩阵图谱
cm = sm.confusion_matrix(test_y, pred_test_y)
print(cm)
mp.figure('Confusion Matrix', facecolor='lightgray')
mp.title('Confusion Matrix', fontsize=20)
mp.xlabel('Predicted Class', fontsize=14)
mp.ylabel('True Class', fontsize=14)
mp.xticks(np.unique(pred_test_y))
mp.yticks(np.unique(test_y))
mp.tick_params(labelsize=10)
mp.imshow(cm, interpolation='nearest', cmap='jet')
mp.show()
```

分类报告

sklearn.metrics提供了分类报告相关API，不仅可以得到混淆矩阵，还可以得到交叉验证查准率、召回率、f1得分的结果，可以方便的分析出哪些样本是异常样本。

```
# 获取分类报告
cr = sm.classification_report(实际输出, 预测输出)
```

案例：输出分类报告：

```
# 获取分类报告
cr = sm.classification_report(test_y, pred_test_y)
print(cr)
```