机器学习DAY05

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	5	0	0
B类别	0	6	0
C类别	0	0	7

上述矩阵即为理想的混淆矩阵。不理想的混淆矩阵如下：

	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

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)
```

决策树分类

决策树分类模型会找到与样本特征匹配的叶子节点然后以投票的方式进行分类。在样本文件中统计了小汽车的常见特征信息及小汽车的分类，使用这些数据基于决策树分类算法训练模型预测小汽车等级。

汽车价格	维修费用	车门数量	载客数	后备箱	安全性	汽车级别

案例：基于决策树分类算法训练模型预测小汽车等级。

1. 读取文本数据，对每列进行标签编码，基于随机森林分类器进行模型训练，进行交叉验证。

```
import numpy as np
import sklearn.preprocessing as sp
import sklearn.ensemble as se
import sklearn.model_selection as ms

data = np.loadtxt('../data/car.txt', delimiter=',', dtype='U10')
data = data.T
encoders = []
train_x, train_y = [],[]
for row in range(len(data)):
    encoder = sp.LabelEncoder()
    if row < len(data) - 1:
        train_x.append(encoder.fit_transform(data[row]))
    else:
```

```
        train_y = encoder.fit_transform(data[row])
    encoders.append(encoder)
train_x = np.array(train_x).T
# 随机森林分类器
model = se.RandomForestClassifier(max_depth=6, n_estimators=200,
random_state=7)
print(ms.cross_val_score(model, train_x, train_y, cv=4,
scoring='f1_weighted').mean())
model.fit(train_x, train_y)
```

1. 自定义测试集，使用已训练的模型对测试集进行测试，输出结果。

```
data = [
    ['high', 'med', '5more', '4', 'big', 'low', 'unacc'],
    ['high', 'high', '4', '4', 'med', 'med', 'acc'],
    ['low', 'low', '2', '4', 'small', 'high', 'good'],
    ['low', 'med', '3', '4', 'med', 'high', 'vgood']]

data = np.array(data).T
test_x, test_y = [],[]
for row in range(len(data)):
    encoder = encoders[row]
    if row < len(data) - 1:
        test_x.append(encoder.transform(data[row]))
    else:
        test_y = encoder.transform(data[row])
test_x = np.array(test_x).T
pred_test_y = model.predict(test_x)
print((pred_test_y == test_y).sum() / pred_test_y.size)
print(encoders[-1].inverse_transform(test_y))
print(encoders[-1].inverse_transform(pred_test_y))
```

验证曲线

验证曲线：模型性能 = f(超参数)

验证曲线所需API：

```
train_scores, test_scores = ms.validation_curve(
    model,                  # 模型
    输入集, 输出集,
    'n_estimators',         #超参数名
    np.arange(50, 550, 50), #超参数序列
    cv=5                    #折叠数
)
```

train_scores的结构:

参数取值	第一次cv	第二次cv	第三次cv	第四次cv	第五次cv
50	0.91823444	0.91968162	0.92619392	0.91244573	0.91040462
100	0.91968162	0.91823444	0.91244573	0.92619392	0.91244573
...	...	...	...	...	...

test_scores的结构与train_scores的结构相同。

案例：在小汽车评级案例中使用验证曲线选择较优参数。

```
# 获得关于n_estimators的验证曲线
model = se.RandomForestClassifier(max_depth=6, random_state=7)
n_estimators = np.arange(50, 550, 50)
train_scores, test_scores = ms.validation_curve(model, train_x, train_y,
'n_estimators', n_estimators, cv=5)
print(train_scores, test_scores)
train_means1 = train_scores.mean(axis=1)
for param, score in zip(n_estimators, train_means1):
    print(param, '->', score)

mp.figure('n_estimators', facecolor='lightgray')
mp.title('n_estimators', fontsize=20)
mp.xlabel('n_estimators', fontsize=14)
mp.ylabel('F1 Score', fontsize=14)
mp.tick_params(labelsize=10)
mp.grid(linestyle=':')
mp.plot(n_estimators, train_means1, 'o-', c='dodgerblue', label='Training')
mp.legend()
mp.show()
```

```
# 获得关于max_depth的验证曲线
model = se.RandomForestClassifier(n_estimators=200, random_state=7)
max_depth = np.arange(1, 11)
train_scores, test_scores = ms.validation_curve(
    model, train_x, train_y, 'max_depth', max_depth, cv=5)
train_means2 = train_scores.mean(axis=1)
for param, score in zip(max_depth, train_means2):
    print(param, '->', score)

mp.figure('max_depth', facecolor='lightgray')
mp.title('max_depth', fontsize=20)
mp.xlabel('max_depth', fontsize=14)
mp.ylabel('F1 Score', fontsize=14)
mp.tick_params(labelsize=10)
mp.grid(linestyle=':')
mp.plot(max_depth, train_means2, 'o-', c='dodgerblue', label='Training')
mp.legend()
mp.show()
```

学习曲线

学习曲线：模型性能 = f(训练集大小)

学习曲线所需API：

```
_, train_scores, test_scores = ms.learning_curve(
    model,      # 模型
    输入集, 输出集,
    train_sizes=[0.9, 0.8, 0.7],    # 训练集大小序列
    cv=5        # 折叠数
)
```

train_scores的结构:

案例：在小汽车评级案例中使用学习曲线选择训练集大小最优参数。

```
# 获得学习曲线
model = se.RandomForestClassifier( max_depth=9, n_estimators=200,
random_state=7)
train_sizes = np.linspace(0.1, 1, 10)
_, train_scores, test_scores = ms.learning_curve(
    model, x, y, train_sizes=train_sizes, cv=5)
test_means = test_scores.mean(axis=1)
for size, score in zip(train_sizes, train_means):
    print(size, '->', score)
mp.figure('Learning Curve', facecolor='lightgray')
mp.title('Learning Curve', fontsize=20)
mp.xlabel('train_size', fontsize=14)
mp.ylabel('F1 Score', fontsize=14)
mp.tick_params(labelsize=10)
mp.grid(linestyle=':')
mp.plot(train_sizes, test_means, 'o-', c='dodgerblue', label='Training')
mp.legend()
mp.show()
```

案例：预测工人工资收入。

读取adult.txt，针对不同形式的特征选择不同类型的编码器，训练模型，预测工人工资收入。

1. 自定义标签编码器，若为数字字符串，则使用该编码器，保留特征数字值的意义。

```
class DigitEncoder():

    def fit_transform(self, y):
        return y.astype(int)

    def transform(self, y):
        return y.astype(int)

    def inverse_transform(self, y):
        return y.astype(str)
```

1. 读取文件，整理样本数据，对样本矩阵中的每一列进行标签编码。

```
num_less, num_more, max_each = 0, 0, 7500
data = []

txt = np.loadtxt('../data/adult.txt', dtype='U20', delimiter=', ')
for row in txt:
    if(' ?' in row):
        continue
    elif(str(row[-1]) == '<=50K'):
        num_less += 1
        data.append(row)
    elif(str(row[-1]) == '>50K'):
        num_more += 1
        data.append(row)

data = np.array(data).T
encoders, x = [], []
```

```
for row in range(len(data)):
    if str(data[row, 0]).isdigit():
        encoder = DigitEncoder()
    else:
        encoder = sp.LabelEncoder()
    if row < len(data) - 1:
        x.append(encoder.fit_transform(data[row]))
    else:
        y = encoder.fit_transform(data[row])
    encoders.append(encoder)
```

1. 划分训练集与测试集，基于朴素贝叶斯分类算法构建学习模型，输出交叉验证分数，验证测试集。

```
x = np.array(x).T
train_x, test_x, train_y, test_y = ms.train_test_split(
    x, y, test_size=0.25, random_state=5)
model = nb.GaussianNB()
print(ms.cross_val_score(
    model, x, y, cv=10, scoring='f1_weighted').mean())
model.fit(train_x, train_y)
pred_test_y = model.predict(test_x)
print((pred_test_y == test_y).sum() / pred_test_y.size)
```

1. 模拟样本数据，预测收入级别。

```
data = [['39', 'State-gov', '77516', 'Bachelors',
         '13', 'Never-married', 'Adm-clerical', 'Not-in-family',
         'White', 'Male', '2174', '0', '40', 'United-States']]
data = np.array(data).T
x = []
for row in range(len(data)):
    encoder = encoders[row]
    x.append(encoder.transform(data[row]))
x = np.array(x).T
pred_y = model.predict(x)
print(encoders[-1].inverse_transform(pred_y))
```

支持向量机(SVM)

支持向量机原理

1. **寻求最优分类边界**

   正确：对大部分样本可以正确地划分类别。

   泛化：最大化支持向量间距。

   公平：与支持向量等距。

   简单：线性，直线或平面，分割超平面。

2. **基于核函数的升维变换**

   通过名为核函数的特征变换，增加新的特征，使得低维度空间中的线性不可分问题变为高维度空间中的线性可分问题。

   线性核函数：linear，不通过核函数进行维度提升，仅在原始维度空间中寻求线性分类边界。

   基于线性核函数的SVM分类相关API：

```
model = svm.SVC(kernel='linear')
model.fit(train_x, train_y)
```

案例：对simple2.txt中的数据进行分类。

```
import numpy as np
import sklearn.model_selection as ms
import sklearn.svm as svm
import sklearn.metrics as sm
import matplotlib.pyplot as mp
x, y = [], []
data = np.loadtxt('../data/multiple2.txt', delimiter=',', dtype='f8')
x = data[:, :-1]
y = data[:, -1]
train_x, test_x, train_y, test_y = \
    ms.train_test_split(x, y, test_size=0.25, random_state=5)
# 基于线性核函数的支持向量机分类器
model = svm.SVC(kernel='linear')
model.fit(train_x, train_y)
n = 500
l, r = x[:, 0].min() - 1, x[:, 0].max() + 1
b, t = x[:, 1].min() - 1, x[:, 1].max() + 1
grid_x = np.meshgrid(np.linspace(l, r, n),
                     np.linspace(b, t, n))
flat_x = np.column_stack((grid_x[0].ravel(), grid_x[1].ravel()))
flat_y = model.predict(flat_x)
grid_y = flat_y.reshape(grid_x[0].shape)
pred_test_y = model.predict(test_x)
cr = sm.classification_report(test_y, pred_test_y)
print(cr)
mp.figure('SVM Linear Classification', facecolor='lightgray')
mp.title('SVM Linear Classification', fontsize=20)
mp.xlabel('x', fontsize=14)
mp.ylabel('y', fontsize=14)
mp.tick_params(labelsize=10)
mp.pcolormesh(grid_x[0], grid_x[1], grid_y, cmap='gray')
mp.scatter(test_x[:, 0], test_x[:, 1], c=test_y, cmap='brg', s=80)
mp.show()
```

多项式核函数：poly，通过多项式函数增加原始样本特征的高次方幂

$$y = x_1 + x_2$$
$$y = x_1^2 + 2x_1x_2 + x_2^2$$
$$y = x_1^3 + 3x_1^2x_2 + 3x_1x_2^2 + x_2^3$$

案例，基于多项式核函数训练sample2.txt中的样本数据。

```
# 基于线性核函数的支持向量机分类器
model = svm.SVC(kernel='poly', degree=3)
model.fit(train_x, train_y)
```

径向基核函数：rbf，通过高斯分布函数增加原始样本特征的分布概率

案例，基于径向基核函数训练sample2.txt中的样本数据。

```python
# 基于径向基核函数的支持向量机分类器
# C：正则强度
# gamma：正态分布曲线的标准差
model = svm.SVC(kernel='rbf', C=600, gamma=0.01)
model.fit(train_x, train_y)
```