实验 7：AdaBoost

实验目的

1. 熟悉scikit-learn中的数据集使用方法
2. 了解弱分类器（单层决策树）的构建过程
3. 实现AdaBoost算法
4. 熟悉scikit-learn中AdaBoost分类器的调用

实验数据

1. scikit-learn中的乳腺癌数据集（二分类）。
2. 使用scikit-learn中的make_classification方法生成分类数据集。

实验步骤

1. 数据集

scikit-learn模块中提供了乳腺癌二分类数据集，该数据集包括569个样本，每个样本有30个特征，目标变量是二分类标签，表示肿瘤是良性（0）还是恶性（1）。读取方式如下：

```
from sklearn.model_selection import train_test_split
breastcancerdata = load_breast_cancer()
X = breastcancerdata.data # (569,30)
y = breastcancerdata.target # (569,)
```

除了乳腺癌数据集，你还可以通过make_classification()方法自动生成你需要的分类数据集。使用示例代码如下（sklearn.datasets.make_classification — scikit-learn 1.4.2 documentation）：

```
X, y = make_classification(n_samples=1000, n_features=20, n_informative=2, n_redundant=0, random_state=7, n_classes=2) # 生成1000个样本，每个样本有20个特征，类别数量为2，其他参数信息见官网文档
```

需要注意的是，乳腺癌或自动生成的数据集标签是0和1，我们进行后面实验需要y的取值为-1或1，需要做一下处理。

数据读取部分的代码我们已经给出，这里对y标签做了处理并使用scikit-learn中的train_test_split函数对样本进行了训练集、测试集的划分。

代码：

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer
breastcancerdata = load_breast_cancer()
X = breastcancerdata.data
y = breastcancerdata.target
```

```
# 创建样本数据
'''X, y = make_classification(n_samples=1000, n_features=20, n_informative=2,
n_redundant=0, random_state=7)'''
y = 2*y-1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

2. 构造弱分类器

AdaBoost（Adaptive Boosting）是一个集成学习算法，它通过结合多个简单的分类器（通常称为弱分类器）来形成一个强大的分类器。

在这部分我们给出了弱分类器（单层决策树）的构造代码。单层决策树是一种简单的分类器，只在一个属性（即特征）上进行决策。它通过选择一个属性和阈值，根据是否大于或小于这个阈值将样本分为两类。

代码：

```
# 对于传入的特征、阈值、不等号，给出分类预测值，用于获取最佳决策树
def stump_classify(X, dimension, thresh_val, thresh_ineq):
    """通过阈值比较对数据进行分类的决策树桩分类器"""
    ret_array = np.ones((np.shape(X)[0], 1))
    if thresh_ineq == 'lt':
        ret_array[X[:, dimension] <= thresh_val] = -1.0
    else:
        ret_array[X[:, dimension] > thresh_val] = -1.0
    return ret_array
```

该部分代码对于传入的特征、阈值、不等号，给出单层决策树的分类预测值，用于构造最佳决策树。最佳决策树指分类效果最好准确率最高的单层决策树。构造最佳决策树需找到最优的特征和阈值组合，过程如下：

1. 遍历数据集的每一个特征。
2. 对于每个特征，根据特征的最小值和最大值确定考虑的阈值范围。
3. 在确定的范围内，按步长尝试不同的阈值，并分别测试阈值分割时小于或大于阈值的情况。
4. 对每种情况（特征、阈值、不等号的组合）计算加权错误率，选出错误率最低的组合，构成一个最佳的单层决策树。

代码：

```
def build_stump(X, y, D):  # 传入特征、标签和权重矩阵
    """构建最佳单层决策树"""
    ......
    num_steps = 10.0 # 用于设定阈值取值数量
    ......
    for i in range(n): # 遍历每一个特征
        range_min = X[:, i].min()
        range_max = X[:, i].max()
        step_size = (range_max - range_min) / num_steps # 计算步长，用于确定阈值
        for j in range(-1, int(num_steps) + 1):
            for inequal in ['lt', 'gt']: # less than/greater than
                thresh_val = (range_min + float(j) * step_size) # 设定阈值
```

```
            predicted_vals = stump_classify(X, i, thresh_val, inequal)
            err_arr = np.ones((m, 1)) # 初始化每个样本的错误率为1，预测正确则改为0
            err_arr[predicted_vals == y] = 0
            weighted_error = np.dot(D.T, err_arr) # 计算加权错误率
            if weighted_error < min_error:
                min_error = weighted_error
                best_class_est = predicted_vals.copy()
                best_stump['dim'] = i # 保存最佳决策树参数（特征、阈值、不等号），使用字典类型。
                best_stump['thresh'] = thresh_val
                best_stump['ineq'] = inequal
    return best_stump, min_error, best_class_est # 返回最佳决策树参数字典、错误率以及预测结果
```

3. 实现AdaBoost

在这部分你需完成AdaBoostClassifier_()函数。AdaBoost算法的核心原理和步骤如下：

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: $\mathcal{D}_1(\boldsymbol{x}) = 1/m$.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $h_t = \mathfrak{L}(D, \mathcal{D}_t)$;
4: $\epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;
5: **if** $\epsilon_t > 0.5$ **then break**
6: $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$;
7: $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
$= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$
8: **end for**

输出： $H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$

1. 初始化等权重D分配给每个样本。
2. 循环进行以下步骤指定次数（或直到错误率为0）：
   A. 使用当前权重D找到最佳单层决策树。
   B. 使用错误率计算该决策树的alpha值。
   C. 根据这个alpha和决策树的预测结果更新样本权重D。
   D. 将当前分类器的结果加入到最终的模型预测中。
3. 累计各个分类器的预测结果，形成最终的模型输出。

代码：

```
class AdaBoostClassifier_(BaseEstimator, ClassifierMixin):
    def __init__(self, num_iters=40):
        self.num_iters = num_iters
        self.classifiers = []
    def fit(self, X, y):
        ......
    def predict(self, X):
        ......
```

AdaBoostClassifier_类继承了scikit-learn中的BaseEstimator和ClassifierMixin，这种继承方式是一种常见的操作做法，可以让自定义分类器能够与scikit-learn的其他工具和流程无缝集成。在__init__()中设定了构造弱分类器的数量num_iters，并使用self.classifiers属性存放每一轮的弱分类器最佳决策树。

我们还参照scikit-learn的接口形式在函数内部定义了fit()和predict()方法。你需要参照上述算法原理补全fit()函数。

代码：

```
class AdaBoostClassifier_(BaseEstimator, ClassifierMixin):
    .....
    def fit(self, X, y):
        y = y.reshape(-1, 1)  # 确保 y 是列向量
        m = X.shape[0]
        D = np.ones((m, 1)) / m # 初始化参数矩阵，每个样本权重均为1/m
        agg_class_est = np.zeros((m, 1)) # 初始化每个样本的预测结果为0
        for i in range(self.num_iters):
            '''code here '''
            best_stump['alpha'] = alpha
            self.classifiers.append(best_stump)
            '''code here '''
            agg_class_est += alpha * class_est
            agg_errors = np.multiply(np.sign(agg_class_est) != y, np.ones((m, 1)))
            error_rate = agg_errors.sum() / m
            if error_rate == 0.0:
                break
        return self
    .....
```

在第一个 '''code here '''中，你需要找到最佳单层决策树，可以调用上部分实现的build_stump()函数。然后利用函数返回的错误率计算该决策树对应的alpha值。第m个决策树的alpha的计算公式如下：

$$\alpha_t = \frac{1}{2} log \frac{1 - e_t}{e_t}$$

其中e_t表示错误率，由build_stump()返回。在第二个 '''code here '''中，你需要根据D_t、alpha和决策树的预测结果更新样本权重D_{t+1}。其中的f(x)指样本的标签，$h_t(x)$表示该决策树给出的预测值，也由build_stump()返回。公式分母中的Z_t是规范化因子，目的是为了使D_{t+1}的所有元素和为1，则Z_t可通过将分子数组各项加和得出。

完成了fit()函数，你就可以通过AdaBoostClassifier_类实例化一个分类器对象，获取数据集后即可进行训练和预测。实例化示例代码如下：

```
clf = AdaBoostClassifier_(num_iters=10)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

4. 调用scikit-learn中的AdaBoost

scikit-learn模块也提供了AdaBoost类的接口，调用方法与支持向量机实验中使用的方法相同，通过类实例化一个分类器对象，即可调用fit()、predict()和score()等方法。

代码：

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier_()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
clf.score((X_test, y_test)
```

你可以使用乳腺癌二分类数据集或者自己构造的数据集，来对比scikit-learn和自己编写的AdaBoost的结果。由于弱分类器构造方式的不同以及数据集的特殊性，可能你编写的算法能够获得更高的准确率。

代码：

```
from sklearn.ensemble import AdaBoostClassifier
# 创建模型
clf1 = AdaBoostClassifier_()
clf2 = AdaBoostClassifier()
clf1.fit(X_train, y_train)
clf2.fit(X_train, y_train)
# 进行预测
y_pred1 = clf1.predict(X_test)
# 评估模型
print("Accuracy of clf1:", accuracy_score(y_test, y_pred1))
print("Accuracy of clf2:", clf2.score(X_test, y_test))
```