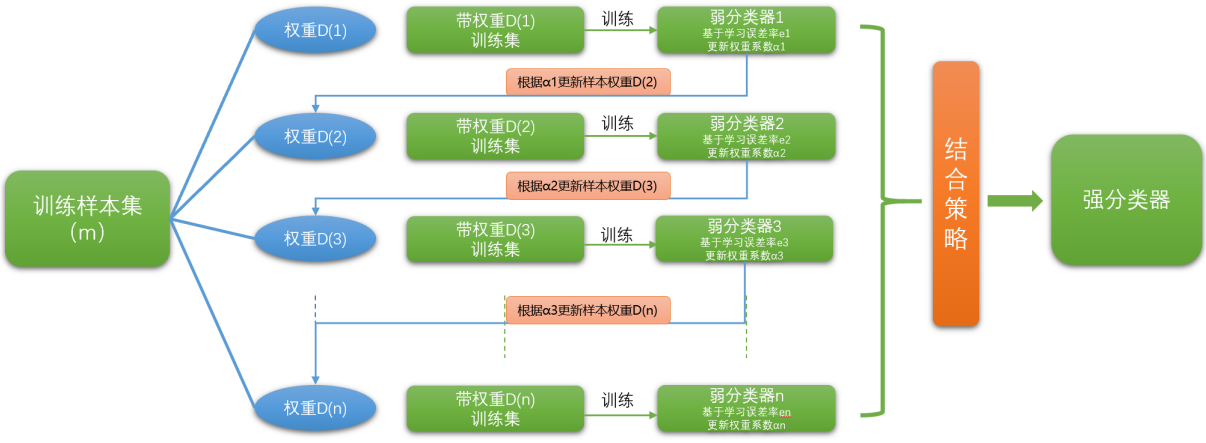
Adaboost算法原理及实现流程【补充】

一、回顾Boosting算法的基本结构

在集成算法的分类中，我们已经讲解了Boosting算法系列的基本思想，如下图：



boosting图示

从图中可以看出，Boosting算法的工作机制是：

1. 将训练集用初始权重训练出一个弱学习器。
2. 根据弱学习器的表现，更新训练样本的权重。从而使得前一个弱学习器的学习误差率高的训练样本的权重增高，从而让这些误差率较高的样本点在后面的弱学习器中得到更多的重视。
3. 然后基于调整权重后的训练集来训练新的弱学习器，如此重复。
4. 当弱学习器数量达到我们指定的数量后，最终将这k个弱学习器通过组合策略整合，得到最终的强学习器。

其中有几个具体问题我们在这里需要继续进行详细说明：

1. 如何计算学习误差率 e？
2. 如何得到弱学习器权重系数 α？
3. 如何更新样本权重 D？
4. 使用何种组合策略?

那么我们就具体看一下Adaboost算法如何解决Boosting家族的这几大问题。

二、Adaboost算法原理

AdaBoost，是英文"Adaptive Boosting"（自适应增强）的缩写，由 Yoav Freund 和 Robert Schapire 在1995年提出。它的自适应在于：前一个基本分类器分错的样本会得到加强（也就是得到更高的权重），加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数，算法停止。

具体说来，整个Adaboost 迭代算法就3步：

- **初始化训练数据的权重。**

  如果有N个样本，则每一个训练样本最开始时都被赋予相同的权值：1/N。

- **训练弱分类器。**

  具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权值就被降低；相反，如果某个样本点没有被准确地分类，那么它的权值就得到提高。然后，权值更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

- **将各个训练得到的弱分类器组合成强分类器。**

  各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

三、Adaboost算法流程

给定一个训练数据集 $T=(x_1,y_1),(x_2,y_2),...(x_N,y_N)$ ，Adaboost的目的就是从训练数据中学习一系列弱分类器或基分类器，然后将这些弱分类器组合成一个强分类器。

1. 初始化训练数据的权重

每一个训练样本最开始时都被赋予相同的权重：$\frac{1}{N}$ ，即

$$D_1 = (w_{11},w_{12},...,w_{1N})$$
$$\text{其中 } w_{1i}=\frac{1}{N},\quad i=1,2,...,N$$

D_1 表示第一个弱分类器的样本权重分布，w_{1i}表示每一个样本的权重大小，i 是当前训练数据集的样本个数。

2. 计算学习误差率

当模型进行多次迭代后，我们用 $k=1,2,...,K$ 表示进行了k 次迭代，则

第一步：使用具有权重分布 D_k 的训练数据集进行学习，得到基本分类器

$$G_k(x):X\to\{-1,+1\}$$

$\{-1,+1\}$ 表示分类型标签的集合，X 表示 x 样本实例的所属实例空间。

第二步：计算 $G_k(x)$ 在训练集上的分类误差率

$$e_k=P(G_k(x_i)\neq y_i)=\sum_{i=1}^{N}w_{ki}I(G_k(x_i)\neq y_i)$$

这里应用了概率论中数学期望的概念：$E=\sum x_ip_i$ 。

由上述式子可知，$G_k(x)$ 在训练数据集上的误差率 e_k 就是被模型 $G_k(x)$ 误分类样本的权值之和。

3. 计算模型 $G_k(x)$ 的权重系数

$$\alpha_k = \frac{1}{2} ln \frac{1-e_k}{e_k}$$

权重系数的推导式大家可以《统计学习方法》第八章进行查看，这里就不再赘述了。

有上述式子可知，当 $e_k \leq \frac{1}{2}$ 时，$\alpha_k \geq 0$ ，且 α_k 随着 e_k 的减小而增大，意味着分类误差率越小的基分类器在最终分类器的作用越大。

4. 更新样本权重 D

$$D_{k+1} = （ w_{k+1,1}, w_{k+1,2}, \ldots, w_{k+1,N})$$
$$w_{k+1,i} = \frac{w_{ki}}{Z_k} exp(-\alpha_k y_i G_k(x_i)), \quad 其中 i = 1, 2, \ldots, N$$

这一步骤的目的是为了得到样本的新的权值分布，并用于下一次迭代。我们将基分类器 $G_k(x)$ 的误分类样本的权重增大，而被正确分类的样本的权重减小。通过这种形式，让Adaboost算法能聚焦于较难分的样本上。

其中 Z_k 称作规范化因子，使得模型 D_{k+1} 成为一个概率分布：

$$Z_k = \sum_{i=1}^{N} w_{ki} exp(-\alpha_k y_i G_k(x_i))$$

从 $w_{k+1,i}$ 计算公式可以看出，如果第 i 个样本分类错误，则 $y_i G_k(x_i) < 0$ ，继而会导致样本的权重在第$k+1$ 个弱分类器中增大；如果分类正确，则权重在第 $k+1$ 个弱分类器中降低。

5. 组合各个弱分类器

$$f(x) = \sum_{k=1}^{K} \alpha_k G_k(x)$$

从而得到最终分类器，如下：

$$G(x) = sign(f(x)) = sign\Big(\sum_{k=1}^{K} \alpha_k G_k(x) \Big)$$

四、Adaboost算法小栗子

我们给定下列训练样本，并用Adaboost算法进行学习。

X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1

求解思路：初始化训练数据的权值分布，令每个权重 $w_1 i = \frac{1}{N} = \frac{1}{10}$，其中 $N = 10$，$i = 1, 2, \ldots, 10$ ，然后进行迭代。

根据X和Y的对应关系，要将这10个样本分成两类，一类是 “1”，另一类是 “-1”。其中 ”0，1，2，6，7，8“ 六个样本对应的类别为 ”1“ ， ”3，4，5，9“ 四个样本对应的类别为 ”-1“。

第一轮迭代：

当 k=1时，权值分布为 D_1 （每个样本的初始化权重为0.1）的训练数据，我们假定三种不同阈值的划分策略，计算可得：

- 特征划分阈值取2.5时，我们可以将数据集划分为两个部分，前半部分是样本 “0，1，2”，后半部分为样本 “3，4，5，6，7，8”。此时当 $x < 2.5$ 时，类别标签取 “1”，$x > 2.5$ 时，类别标签取 “-1” ，则样本 ”6，7，8“ 就会判断错误，该划分方法的误差率为0.3。
- 特征划分阈值取5.5时，我们同样可以将数据集划分为两个部分，前半部分是样本 “0，1，2，3，4，5”，后半部分为样本 “6，7，8”。此时当 $x < 5.5$ 时，类别标签取 “1”，$x > 5.5$ 时，类别标签取 “-1” ，则样本 ”3，4，5，6，7，8“ 六个样本都会被判断错误，该划分方法的误差率为0.6，高于0.5，不可取！故我们令 $x > 5.5$ 时，类别标签取 “1”，$x < 5.5$ 时，类别标签取 “-1” ，此时 “0，1，2，9” 判断错误，误差率为0.4。
- 特征划分阈值取8.5时，我们再将数据集划分为两个部分。此时当 $x < 8.5$ 时，类别标签取 “1”，$x > 8.5$ 时，类别标签取 “-1” ，则样本 ”3，4，5“ 就会判断错误，该划分方法的误差率为0.3。

这里无论阈值是2.5还是8.5，我们分类误差率都仅有0.3，所以我们任意选择一个作为划分依据即可，比如我们选择2.5的阈值构建我们的基分类器：

$$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$

从而得 $G_1(x)$ 在训练数据集上的误差率，即被 $G_1(x)$ 模型错误分类的样本 “6，7，8” 的权重之和：

$$e_1 = P(G_1(x_i) \neq y_i = 3 * 0.1 = 0.3$$

然后根据误差率 e_1 计算 G_1 的权重系数：

$$\alpha_1 = \frac{1}{2} ln \frac{1 - e_1}{e_1} = 0.4236$$

这里的 α_1 代表 $G_1(x)$ 在最终的分类函数中所占的权重，为0.4236.

接着更新训练数据的权值分布，以便用于下一次迭代：

$$D_{k+1} = （ w_{k+1,1}, w_{k+1,2}, \ldots, w_{k+1,N})$$

$$w_{k+1,i} = \frac{w_{ki}}{Z_k} exp(-\alpha_k y_i G_k(x_i)), \quad \text{其中} \ i = 1, 2, \ldots, N$$

讲数据代入到公式中进行计算，最后可得到各个数据样本新的权值分布：
$D_2 = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715)$ 。由此可以看出，因为样本中数据 “6，7，8” 被模型 $G_1(x)$ 分错了，所以它们的权重由初始的0.1增大到了0.1666，反之，其他被分类正确的数据的权重就由0.1减小至了0.0715.

此时，我们的分类函数 $f_1(x) = \alpha_1 \times G_1(x) = 0.4236 G_1(x)$，最终得到第一个基分类器 $sign(f_1(x))$ 在训练数据上有三个误分类点。

从上述第一轮迭代过程可以看出：

- 被误分类样本的权重之和影响误差率
- 误差率影响基分类器在最终分类器中所占的权重

第二轮迭代：

当 k=2时，权值分布为
$D_2 = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715)$ 的训练数据，我们仍然假定三种不同阈值的划分策略，计算可得：

- 特征划分阈值取2.5时，我们可以将数据集划分为两个部分，前半部分是样本 “0，1，2”，后半部分为样本 “3，4，5，6，7，8”。此时当 $x < 2.5$ 时，类别标签取 “1”，$x > 2.5$ 时，类别标签取 “-1” ，则样本 ”6，7，8“ 就会判断错误，该划分方法的误差率为0.1666*3。
- 特征划分阈值取5.5时，我们同样可以将数据集划分为两个部分，前半部分是样本 “0，1，2，3，4，5”，后半部分为样本 “6，7，8”。当 $x > 5.5$ 时，类别标签取 “1”，$x < 5.5$ 时，类别标签取 “-1” ，此时 “0，1，2，9” 判断错误，误差率为0.0715*4。
- 特征划分阈值取8.5时，我们再将数据集划分为两个部分。此时当 $x < 8.5$ 时，类别标签取 “1”，$x > 8.5$ 时，类别标签取 “-1” ，则样本 ”3，4，5“ 就会判断错误，该划分方法的误差率为0.0715*3。

所以，阈值取8.5时误差率最低，我们选择8.5的阈值构建我们的基分类器：

$$G_2(x) = \begin{cases} 1, & x < 8.5 \\ -1, & x > 8.5 \end{cases}$$

此时，样本 ”3，4，5“ 就会判断错误，根据 D_2 可知，三个样本的权重分别为0.0715，0.0715，0.0715，所以 $G_2(x)$ 模型在训练集的误差率 $e_2 = P(G_2(x_i) \neq y_i) = 0.0715 \times 3 = 0.2143$.

然后继续计算 G_2 的权重系数：

$$\alpha_1 = \frac{1}{2} ln \frac{1 - e_1}{e_1} = 0.6496$$

更新训练数据的权值分布：

$$D_{k+1} = （ w_{k+1,1}, w_{k+1,2}, \ldots, w_{k+1,N})$$
$$w_{k+1,i} = \frac{w_{ki}}{Z_k} exp(-\alpha_k y_i G_k(x_i)), \quad 其中 i = 1, 2, \ldots, N$$

最终计算得到
$D_3 = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.1667, 0.1060, 0.1060, 0.1060, 0.0455)$. 其中被分错的样本 “3，4，5” 的权重变大， 其他分类的正确的样本权重相应变小。

此时，我们的分类函数 $f_2(x) = 0.4236G_1(x) + 0.6496G_2(x)$，最终得到第二个基分类器 $sign(f_2(x))$ 在训练数据上有三个误分类点。

第三轮迭代：

当 k=3时，权值分布为
$D_3 = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.1667, 0.1060, 0.1060, 0.1060, 0.0455)$ 的训练数据，我们仍然假定三种不同阈值的划分策略，计算可得：

- 特征划分阈值取2.5时，我们可以将数据集划分为两个部分，前半部分是样本 “0，1，2”，后半部分为样本 “3，4，5，6，7，8”。此时当 $x < 2.5$ 时，类别标签取 “1”，$x > 2.5$ 时，类别标签取 “-1” ，则样本 ”6，7，8“ 就会判断错误，该划分方法的误差率为0.1060*3。
- 特征划分阈值取5.5时，我们同样可以将数据集划分为两个部分，前半部分是样本 “0，1，2，3，4，5”，后半部分为样本 “6，7，8”。当 $x > 5.5$ 时，类别标签取 “1”，$x < 5.5$ 时，类别标签取 “-1” ，此时 “0，1，2，9” 判断错误，误差率为0.0455*4。
- 特征划分阈值取8.5时，我们再将数据集划分为两个部分。此时当 $x < 8.5$ 时，类别标签取 “1”，$x > 8.5$ 时，类别标签取 “-1” ，则样本 ”3，4，5“ 就会判断错误，该划分方法的误差率为0.1667*3。

所以，阈值取8.5时误差率最低，我们选择8.5的阈值构建我们的基分类器：

$$G_3(x) = \begin{cases} -1, & x < 5.5 \\ 1, & x > 5.5 \end{cases}$$

此时，样本 ”0，1，2，9“ 就会判断错误，根据 D_3 可知，四个样本权重均为0.0455，所以 $G_3(x)$ 模型在训练集的误差率 $e_3 = P(G_3(x_i) \neq y_i) = 0.0455 \times 34 = 0.1820$.

然后继续计算 G_3 的权重系数：

$$\alpha_1 = \frac{1}{2} ln \frac{1 - e_1}{e_1} = 0.7514$$

更新训练数据的权值分布：

$$D_{k+1} = （ w_{k+1,1}, w_{k+1,2}, \ldots, w_{k+1,N})$$
$$w_{k+1,i} = \frac{w_{ki}}{Z_k} exp(-\alpha_k y_i G_k(x_i)), \quad 其中 i = 1, 2, \ldots, N$$

最终计算得到 $D_4 = (0.125，0.125，0.125，0.102，0.102，0.102，0.065，0.065，0.065，0.125)$. 其中被分错的样本 “0，1，2，9” 的权重变大， 其他分类的正确的样本权重相应变小。

此时，我们的分类函数 $f_3(x) = 0.4236G_1(x) + 0.6496G_2(x) + 0.7514G_3(x)$，至此，第三个基分类器 $sign(f_3(x))$ 在训练数据上有0个错分类点，整个迭代过程就结束了。（并且经此过程，所有的误分类点的权重都得到了平衡化）

最终分类器 $G(x)=sign(f_3(x))=sign[\alpha_1 \times G_1(x)+\alpha_2 \times G_2(x)+\alpha_3 \times G_3(x)]$，讲刚才所计算的 α 的取值分别代入式中，得到最终的分类器为：
$G(x)=sign(f_3(x))=sign[0.4236G_1(x)+0.6496G_2(x)+0.7514G_3(x)]$

五、手写Adaboost算法

Adaboost的步骤我们已经介绍过了，但是为什么要使用这些函数，还是没有介绍过，这里我们也不详细的介绍了。这里需要注意的是对于Adaboost来说有一个很巧妙的解法，这个解法是后人发现的，而创造adaboost算法的人没有察觉到。这是一个1/2的法则，意思就是上一次迭代后预测错误的样本的权重，在下一次这些样本的权重和肯定是1/2, 被预测正确的样本在，下一次跌倒后这样预测正确的样本的权重和也肯定是1/2。为什么是1/2呢？这个是需要根据Adaboost的原理来证明的。

这个理论的好处是，在计算的过程中，再也不用计算对数和指数了。只需要将上一次预测错误的样本的权重求和，然后尝试放大到1/2，计算出放大的值，再使用错误的样本 * 这个算出的值就可以得到新的权重了。

我们这里介绍如何手写adaboost，整个过程也不是很难。为了简便，我们的每一个基分类器还是决策树。

```
# 开始手写adaboost, 使用1/2巧妙写法完成算法

# 设置一些基分类器的参数值
n_estimatros = 100
criterion = 'gini'
max_depth = 5
random_state = 100
max_features = 'auto'

# 设置一些随机数种子，制造一个随机数种子list，用于每一个基分类器
#np.random.seed(random_state)
random_seed_list = (np.random.rand(n_estimatros) * n_estimatros *
10).astype(int)

# 强制转变Y的值，这个是因为adaboost在计算的过程中，必须将label强制的变成1和-1
Y[Y == 0] = -1

import copy

# 制造一个base_estimator
base_estimator = DecisionTreeClassifier(criterion = criterion,
                                        max_depth = 6,
                                        max_features = max_features
                                       )

# 首先初始化权重，并且将其间的权重，误差率，模型系数，都记录下来
weight = np.array([1/X.shape[0]] * X.shape[0])
weight_list = [weight.copy()]
error_list = []
coef_list = []
```

```
estimators_ = []

for i in random_seed_list:
    tree = copy.deepcopy(base_estimator)
    tree.random_state = i

    # 基分类器进行训练
    tree.fit(X, Y, sample_weight = weight)

    # 将tree加入到estimators_
    estimators_.append(tree)

    # 得出当前模型的误差率
    error_rate = 1 - tree.score(X, Y, sample_weight = weight)
    ada_coef_ = 0.5 * np.log((1 - error_rate) / error_rate)

    # error_rate 和 coef加入list
    error_list.append(error_rate)
    coef_list.append(ada_coef_)

    # 使用巧妙1/2法则，权重做更改

    Sample_wrong = (tree.predict(X) != Y)

    coef_wrong = 0.5 / weight[Sample_wrong].sum()
    weight[Sample_wrong] = weight[Sample_wrong] * coef_wrong
    coef_right = 0.5 / weight[~Sample_wrong].sum()
    weight[~Sample_wrong] = weight[~Sample_wrong] * coef_right

    weight_list.append(weight.copy())

# 开始predict
Xtest = X.copy()
Ytest = Y.copy()

# 将每一个样本的所对应的ai * gi(Xtest)算出来，并且做一个求和
predict_result = np.array([a * model.predict(Xtest) for a, model in
zip(coef_list, estimators_)])
predict_rough = pd.Series(predict_result.sum(axis = 0))

# 判断每一个数，如果是小于0，判别为-1，否则判别为1
Y_pred = predict_rough.apply(lambda i : -1 if i < 0 else 1).values

print(accuracy_score(Y_pred, Y))
```

将上面的步骤全部写成函数

```
# 将上面的adaboost写成函数
```

```
import copy

def frank_ada(X, Y, base_estimator,
              n_estimatros = 100,
              random_state = None,
             ):
    if random_state is not None:
        np.random.seed(random_state)
    random_seed_list = (np.random.rand(n_estimatros) * n_estimatros *
1000).astype(int)

    weight = np.array([1 / X.shape[0]] * X.shape[0])
    weight_list = [weight.copy()]
    error_list = []
    coef_list = []
    estimators_ = []

    for i in random_seed_list:
        tree = copy.deepcopy(base_estimator)
        tree.random_state = i
        tree.fit(X, Y, sample_weight = weight)
        estimators_.append(tree)
        error_rate = 1 - tree.score(X, Y, sample_weight = weight)
        ada_coef_ = 0.5 * np.log((1 - error_rate) / error_rate)

        Sample_wrong = (tree.predict(X) != Y)

        coef_wrong = 0.5 / weight[Sample_wrong].sum()
        weight[Sample_wrong] = weight[Sample_wrong] * coef_wrong
        coef_right = 0.5 / weight[~Sample_wrong].sum()
        weight[~Sample_wrong] = weight[~Sample_wrong] * coef_right

        weight_list.append(weight.copy())
        error_list.append(error_rate)
        coef_list.append(ada_coef_)

    return estimators_, weight_list, error_list, coef_list

def ada_predict(X, Y, estimators_):
    predict_result = np.array([coef * model.predict(X) for coef, model in
zip(coef_list, estimators_)])
    predict_rough = predict_result.sum(axis = 0)
    Y_pred = pd.Series(predict_rough).apply(lambda i : 1 if i >= 0 else
-1).values
    score = accuracy_score(Y_pred, Y)
    return Y_pred, score
```

函数构造好之后可以尝试测试代码

```
from sklearn.model_selection import train_test_split

X, Y = make_classification(n_samples=1000, n_features=30,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)

Y[Y==0] = -1

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size = 0.3,
random_state = 108)

base_estimator = DecisionTreeClassifier(criterion = 'gini',
                                        max_depth = 6,
                                        max_features = 'auto'
                                       )
estimators_, weight_list, error_list, coef_list = frank_ada(Xtrain, Ytrain,
                                                            base_estimator,
random_state = 420,
                                                            n_estimatros = 50
                                                           )

Y_pred, score_train = ada_predict(Xtrain, Ytrain, estimators_)
Y_pred, score_test = ada_predict(Xtest, Ytest, estimators_)
print(score_train, score_test)

# 与sklearn的adaboost算法做比较
DTC = DecisionTreeClassifier(max_depth = 6, criterion = 'gini')
ada = AdaBoostClassifier(base_estimator = DTC, n_estimators = 100).fit(Xtrain,
Ytrain)
print(ada.score(Xtrain, Ytrain), ada.score(Xtest, Ytest))
```

最后的结果可以看出Adaboost是有过拟合的情况，并且我们写出的函数结果也与sklearn的Adaboost结果非常相近，算法手写成功。