AdaBoost

1 AdaBoost

1.1 提升方法

提升算法是从弱学习算法出发，反复学习，得到一系列弱分类器（也称基本分类器），然后组合这些弱分类器，构成一个强分类器。其中最重要的两个问题是：

- 如何改变训练数据的权值或概率分布
- 如何将弱分类器组合成强分类器

1.2 AdaBoost 算法

AdaBoost 是一种提升算法。在训练时，AdaBoost 提高那些被前一轮弱分类器错误分类样本的权值，以改变训练数据的权值；此外，AdaBoost 采取加权多数表决的方法，即加大分类误差率小的弱分类器的权值。

给定数据集

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$$

其中，$x_i \in R^n, y_i \in \{-1, 1\}, i = 1, 2, ..., N$。AdaBoost 算法如下：

输入：数据集 $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$，其中，$x_i \in R^n, y_i \in \{-1, 1\}, i = 1, 2, ..., N$，弱分类器个数 M **和**弱学习算法。

1. 初始化训练数据的权值分布

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

2. 对 $m = 1, 2, ..., M$:

- 使用具有权值分布 D_m 的训练数据集学习，得到基本分类器 $G_m(x) : X \to \{-1, 1\}$
- 计算 $G_m(x)$ 在训练集数据上的分类误差率

$$e_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

- 计算 $G_m(x)$ 的系数

$$\alpha_m = \frac{1}{2}\ln\frac{1-e_m}{e_m}$$

- 更新训练集数据的权值分布

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, ..., w_{m+1,N})$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m}\exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, ..., N$$

其中，Z_m 是规范化因子：$Z_m = \sum_{i=1}^{N}\exp(-\alpha_m y_i G_m(x_i))$

3. 构建基本分类器的线性组合 $f(x) = \sum_{m=1}^{M}\alpha_m G_m(x)$，得到最终分类器

$$G(x) = sign(f(x)) = sign(\sum_{m=1}^{M}\alpha_m G_m(x))$$

输出：分类器 $G(x)$。

1.3 训练误差分析

AdaBoost 算法的最终训练误差为 $\frac{1}{N}\sum_{i=1}^{N} I(G(x_i) \neq y_i)$。由于当 $G(x_i) \neq y_i$ 时，$y_i f(x_i) < 0$，因此 $\exp(-y_i f(x_i)) \geq 1$，因此有：

$$\begin{aligned}
\frac{1}{N}\sum_{i=1}^{N} I(G(x_i) \neq y_i) &\leq \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i f(x_i)) \\
&= \frac{1}{N}\sum_{i=1}^{N}\exp(-\sum_{m=1}^{M}\alpha_i y_i G_m(X_i)) \\
&= \sum_{i=1}^{N} w_{1i}\prod_{m=1}^{M}\exp(-\alpha_i y_i G_m(X_i)) \\
&= Z_1\sum_{i=1}^{N} w_{2i}\prod_{m=2}^{M}\exp(-\alpha_i y_i G_m(X_i)) \\
&= Z_1 Z_2\sum_{i=1}^{N} w_{3i}\prod_{m=3}^{M}\exp(-\alpha_i y_i G_m(X_i)) \\
&= ... \\
&= Z_1 Z_2 ... Z_{M-1}\sum_{i=1}^{N} w_{Mi}\exp(-\alpha_M y_i G_M(X_i)) \\
&= \prod_{m=1}^{M} Z_m
\end{aligned}$$

这说明，在每一轮可以选取适当的 G_m 使得 Z_m 最小，从而使训练误差下降最快。此外，由于：

$$
\begin{aligned}
Z_m &= \sum_{i=1}^{N} w_{mi} \exp(-\alpha_i y_i G_m(X_i)) \\
&= \sum_{y_i = G_m(x_i)} w_{mi} e^{-\alpha_m} + \sum_{y_i \neq G_m(x_i)} w_{mi} e^{\alpha_m} \\
&= (1 - e_m) e^{-\alpha_m} + e_m e^{\alpha_m}
\end{aligned}
$$

将 $\alpha_m = \frac{1}{2} \ln \frac{1-e_m}{e_m}$ 代入，有：

$$
Z_m = 2\sqrt{e_m(1-e_m)} = \sqrt{1 - 4\gamma_m^2} \leq \exp(-2\gamma_m^2)
$$

其中 $\gamma_m = \frac{1}{2} - e_m$。式子中的不等式可以由 $\sqrt{1-x} \leq e^x, x \geq 0$ 得出。所以若存在 $\gamma > 0$，对所有 m 有 $\gamma_m \geq \gamma$，则有：

$$
\frac{1}{N} \sum_{i=1}^{N} I(G(x_i) \neq y_i) = \prod_{m=1}^{M} Z_m \leq \exp(-2 \sum_{m=1}^{M} \gamma_m^2) \leq \exp(-2M\gamma^2)
$$

因此，AdaBoost 的训练误差是以指数速率下降的。

2 代码实现

考虑以下数据：

x	y
0	1
1	1
2	1
3	-1
4	-1
5	-1
6	1
7	1
8	1
9	-1

2.1 sklearn 实现

准备数据：

```
import numpy as np
X = np.array([i for i in range(10)]).reshape(-1,1)
Y = np.array([1,1,1,-1,-1,-1,1,1,1,-1])
```

AdaBoost 的 sklearn 接口位于 sklearn.ensemble.AdaBoostClassifier, 其文档可见**此处**。其中较为重要的参数有：

- base_estimator：基分类器，默认为深度为 1 的决策树
- n_estimators：基分类器的个数，默认为 50

```
from sklearn.ensemble import AdaBoostClassifier
clf = [AdaBoostClassifier(n_estimators=i+1) for i in range(4)]
for c in clf:
    c.fit(X,Y)
```

```
for c in clf:
    print('基分类器个数：',clf.index(c)+1,'准确率：',np.sum(c.predict(X)==Y)/len(Y))
```

```
## 基分类器个数： 1 准确率： 0.7
## 基分类器个数： 2 准确率： 0.9
## 基分类器个数： 3 准确率： 1.0
## 基分类器个数： 4 准确率： 1.0
```

2.2 AdaBoost 实现

此处 AdaBoost 也使用深度为 1 的决策树作为基分类器。

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
X = np.array([i for i in range(10)]).reshape(-1,1)
Y = np.array([1,1,1,-1,-1,-1,1,1,1,-1]).reshape(-1,1)
```

AdaBoost 类如下：

```
class AdaBoost:

    def __init__(self,n_estimators=10):
        self.n_estimators = n_estimators
```

```
    def fit(self,X,Y):
        self.clf_list = []
        self.alpha_list = []

        weight = [1/len(Y) for _ in range(len(Y))]
        df = pd.concat([pd.DataFrame(X),pd.DataFrame(Y,columns=['y']),\
                        pd.DataFrame({'weight':weight})],axis=1)

        for m in range(self.n_estimators):
            X = df.iloc[:,:-2].values
            Y = df.iloc[:,-2].values
            weight = df.iloc[:,-1].values.reshape(-1)

            clf = DecisionTreeClassifier(max_depth=1)
            clf.fit(X,Y,sample_weight = weight)

            df['predict'] = list(clf.predict(X))
            err = np.sum(df['predict']!=df['y'])/len(df)
            alpha = np.log((1-err)/err)/2

            df['weight'] = df['weight']*np.exp(-alpha*df['y']*df['predict'])
            df['weight'] = df['weight']/np.sum(df['weight'])
            del df['predict']

            self.clf_list.append(clf)
            self.alpha_list.append(alpha)

    def predict(self,new_X):
        tmp = np.array([clf.predict(new_X) for clf in self.clf_list])
        tmp = tmp*np.array(self.alpha_list).reshape(-1,1)
        return np.sign(np.sum(tmp,axis=0))
```

预测结果如下，可以看到自写的 AdaBoost 分类器在稳定性上略逊一筹：

```
clf = [AdaBoost(n_estimators=i+1) for i in range(10)]
for c in clf:
    c.fit(X,Y)
    print('基分类器个数: ',clf.index(c)+1,'准确率: ',\
      np.sum(c.predict(X)==Y.reshape(-1))/len(Y))
```

```
## 基分类器个数： 1 准确率： 0.7
## 基分类器个数： 2 准确率： 0.4
## 基分类器个数： 3 准确率： 1.0
## 基分类器个数： 4 准确率： 0.7
## 基分类器个数： 5 准确率： 0.7
## 基分类器个数： 6 准确率： 1.0
## 基分类器个数： 7 准确率： 1.0
## 基分类器个数： 8 准确率： 1.0
## 基分类器个数： 9 准确率： 1.0
## 基分类器个数： 10 准确率： 1.0
```