火龙果 · 整理
uml.org.cn

机 器 学 习 课 程 结 课 论 文

学号、专业：

姓　　名：

论 文 题 目：Adaboost 算法分析及简单应用

指 导 教 师：

所 属 学 院：电子工程与自动化学院

成绩评定	
教师签名	

年　　月　　日

火龙果 · 整理
uml.org.cn

Adaboost 算法分析及简单应用

摘 要: 本文主要阐述了在数据挖掘领域中十个主要的分类算法之一—— Adaboost 的来源、发展，以及开发应用，然后介绍了在该算法的训练过程中一个简单的应用，最后对该算法进行了简单评价。

关键词 : Adaboost 算法；发展背景；训练过程；性能改进；分类算法

Analysis of the algorithm and its simple application

Abstract: This paper mainly describes one of the ten main classification algorithm in datea mining —— Adaboost. Firstly, it introduces the origin, development and application, then introduces the main training process of the algorithm, simple application and finally discusses the algorithm simply.

Keywords: Adaboost algorithm; development background; training process; performance improvement; classi?cation algorithm

1 引言

在 1990 年，Schapire 提出了 Boosting 算法，次年 Freund 改进 Boosting 算法，这两种算法存在共同的实践上的缺陷，那就是都要求事先知道弱学习算法学习正确率的下限。1995 年，Freund 和 Schapire 共同改进了 Boosting 算法，提出了 Adaboost(Adaptive Boosting) 算法，该算法效率和 Freund 于 1991 年提出的 Boosting 算法几乎相同，但不需要任何关于弱学习器的先验知识，因而更容易应用到实际问题当中。

Adaboost 即 Adaptive Boosting，它通过自适应学习算法来降低误差率，多次迭代后达到预期的效果。在另一方面，它并不需要知道样本空间的精确分布，每个样品经过弱学习调节后，通过权重的高低更新空间分布。该算法可以很容易地应用到实际问题，因此，已成为最流行的 Boosting 算法。

在机器学习的算法中，Adaboost 算法是一种比较重要且通用的用于特征分类的算法，在图像检索和人脸表情识别等问题中都有普遍应用。从现状看，人们对 Adaboost 算法的研究及应用主要集中用于分类问题上，另外在某些回归问题上也有所涉及，比如两类问题、多类单标签问题、多类多标签问题和回归问题。该学习算法可以提高其他算法的性能，其思想来自于 1984 年 Valiant 提出的 PAC(Probably Approximately Correct) （可能近似正确）学习模型，在这个 PAC 模型中，提出了两个概念 - 弱学习算法和强学习算法。其概念是 : 如果一个学习算法通过学习一组样本，识别率很高，则称其为强学习算法 ; 如果识别正确率只有 50%左右，仅略高于随机猜测，则称其为弱学习算法。强分类器是指一种算法通过一些训练集的学习，来达到所需的正确识别率。如果一个学习算法的识别率仅好于随机的猜测，则称其为弱分类器。通常，针对一个具体的识别问题，我们很难找到一个理想的强分类器，但是弱分类器一般都会有很多，基于这种现象，Adaboost 算法被提出，它指出：通过一定的算法可以将一组弱分类器提升为一个强分类器。

Adaboost 在机器学习领域中十分重要，它是一种提高任意给定学习算法准确度的方法。也就是说，Adaboost 算法为其他算法提供了一种框架结构，而其他算法只在其中作为子分类器，因此 Adaboost 算法可以运用在许多方面的实践上。如今通过 Adaboost 算法，我们实现了手写体字符识别，运用到了许多输入设备上，如流行的触屏手机上的手写输入、笔记本电脑的手写输入、扫描仪扫面文字转化为电子文档。我们实现了图像识别，如人脸识别、google 上的图片检索功能（很方便地找到我想要的清晰度更高的图片）。我们实现了语音识别，现在的 WIN7 上都有了一个语音识别的功能，我们能够让计算机知道我们说了什么话，并通过机器自动学习提高语音识别的精度，声音输入和声控很可能在未来普及。当然 Adaboost 算法还能做文本分类和医疗诊断等等。

2 算法描述

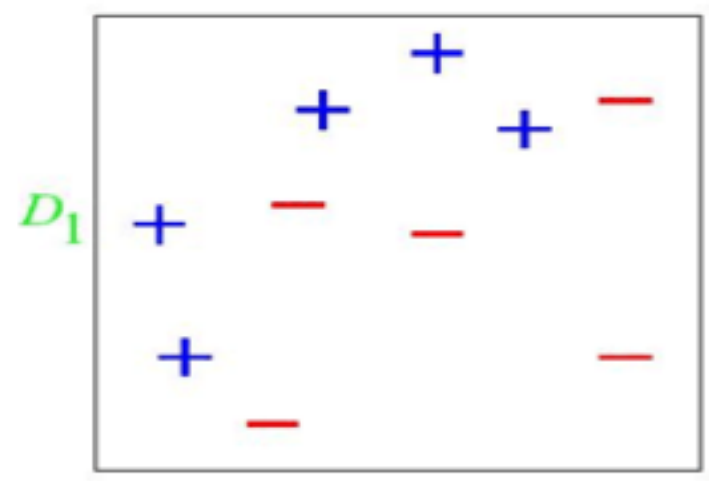
Adaboost 的是一种迭代算法，其主要思想是针对相同的训练集用不同的分类器（弱分类器）去学习，然后把这些弱分类器结合起来，以形成更强的最终分类器（强分类器）。算法本身是通过改变数据分布，根据每个训练的样本的分类是否正确，以及整体分类的精度来确定每个样本的权重来实现的。每次都是将样本加权后得到一组新的数据并将其分给下一层的分类器进行训练，直至达到一定的正确率，然后将这些分类器融合在一起形成一个最终分类器，作为一个最后的决定分类。

Adaboost 算法的基本步骤是：首先，给出一个弱学习算法和训练集 （X1, Y1),

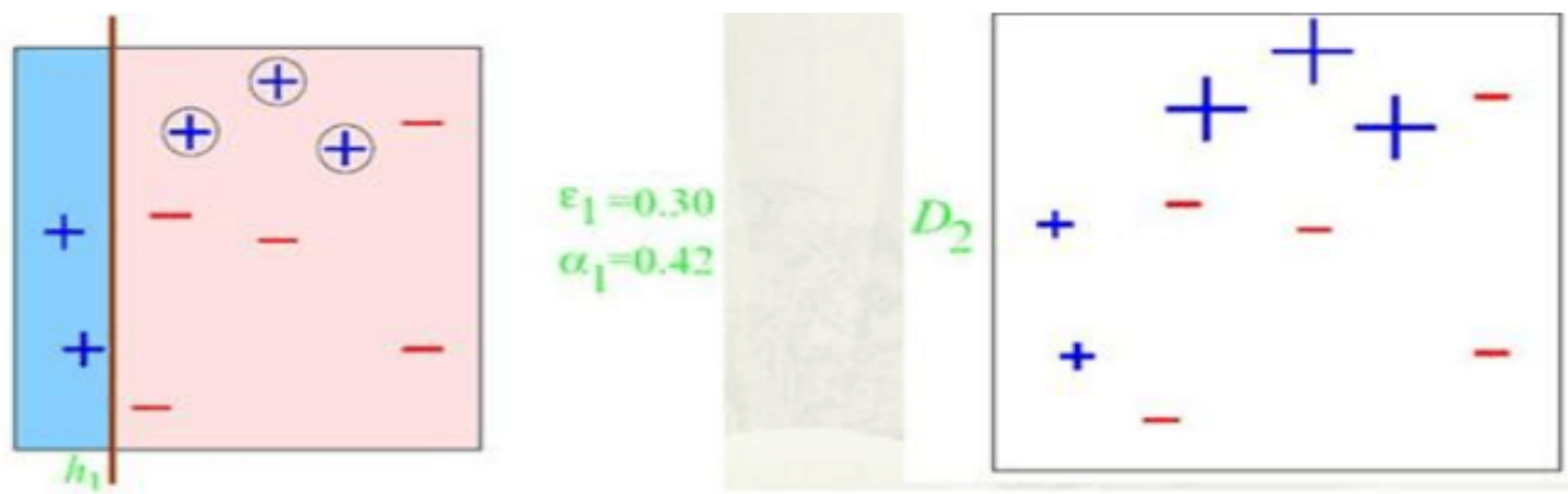
（X2, Y2），...，（XM, YM），其中，XM∈X，X 代表分类问题或类域实例空间，YM∈Y = {+1，-1}。初始化时，Adaboost 算法将训练集指定分配 1/M，即每个训练样本的权重都是 1/M。然后，用弱学习算法进行 T 次迭代，每次迭代都根据训练结果来更新训练集的分布情况，并为训练失败的训练实例给予更大的权重，这样的训练实例将在下一次迭代得到更多的关注。这样经 T 次循环后将会得到 T 个弱分类器，最终按更新的权重叠加到一起得到强分类器，这个最终的强分类器是这些弱分类器的加权平均。虽然单个弱学习器的准确率不高，但采用 Adaboost 算法后，将会使得结果更为准确。Adaboost 算法与 Boosting 算法不同，它不需要预先知道弱分类器分错的概率，并且最后它是通过所有弱分类器的分类精度来得到这个强分类器的分类精度，这样可以提高分类能力。

简单地说，Adaboost 算法的训练是通过调整每个样品相应的权重来实现的。首先，将各样品的权重比设成一样，其中 M是一个弱学习的样本分布的训练次数。对于错误分类的样本，增加相应的权重，而对于正确分类的样本，降低其权重，这样错的样本就受到重视以便下次“对症下药”。在新的样本分布中，又一次进行弱学习算法地训练。这样经 T 周期的循环得到 T 个弱分类器，再把这些弱分类器加权起来，最终得到一个合适的的强分类器。

下面我们举一个简单的例子来看看 Adaboost 的实现过程：



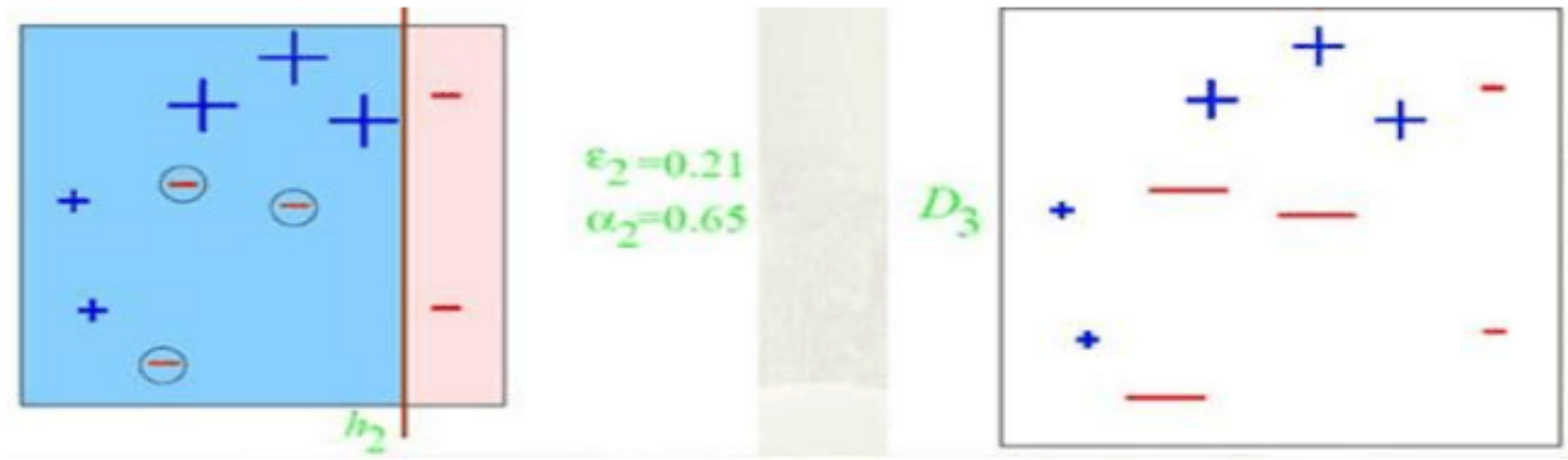
图中，“+”和“-”分别表示两种类别，在这个过程中，我们使用水平或者垂直的直线作为分类器，来进行分类。



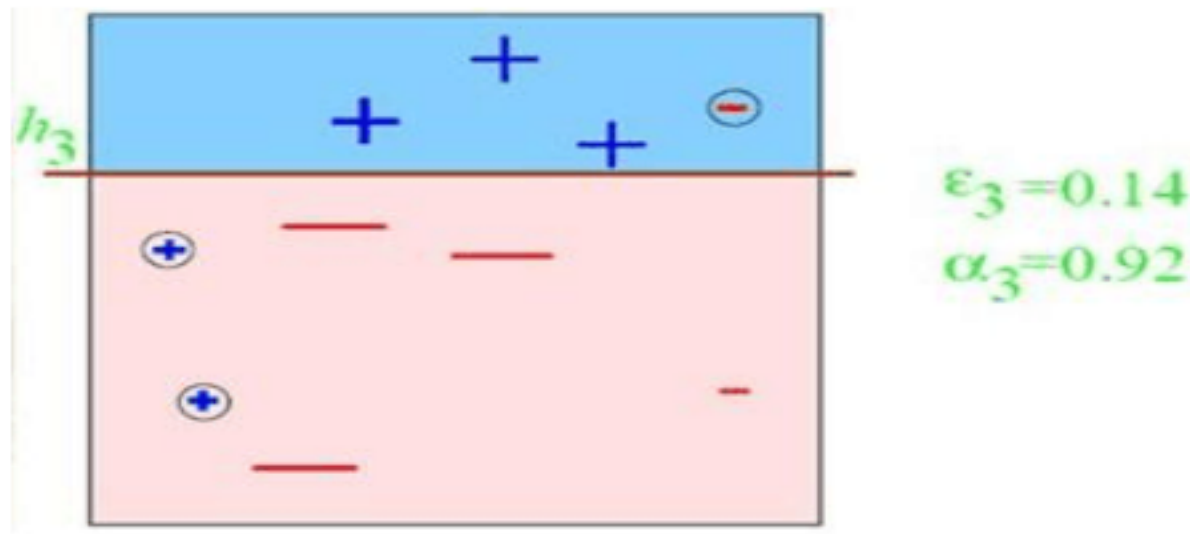
根据分类的正确率，得到一个新的样本分布 D2，一个子分类器 h1。其中划圈的样本表示被分错的。在右边的图中，比较大的“+”表示对该样本做了加权。

算法最开始给了一个均匀分布 D。所以 h1 里的每个点的值是 0.1。当划分后，有三个点划分错了，根据算法误差表达式 $\varepsilon_t = \Pr_{i\sim D_t}[h_t(x_i) \neq y_i]$ 得到误差为分错了的三个点的值之和，所以 ?1=(0.1+0.1+0.1)=0.3，而 α1根据表达式

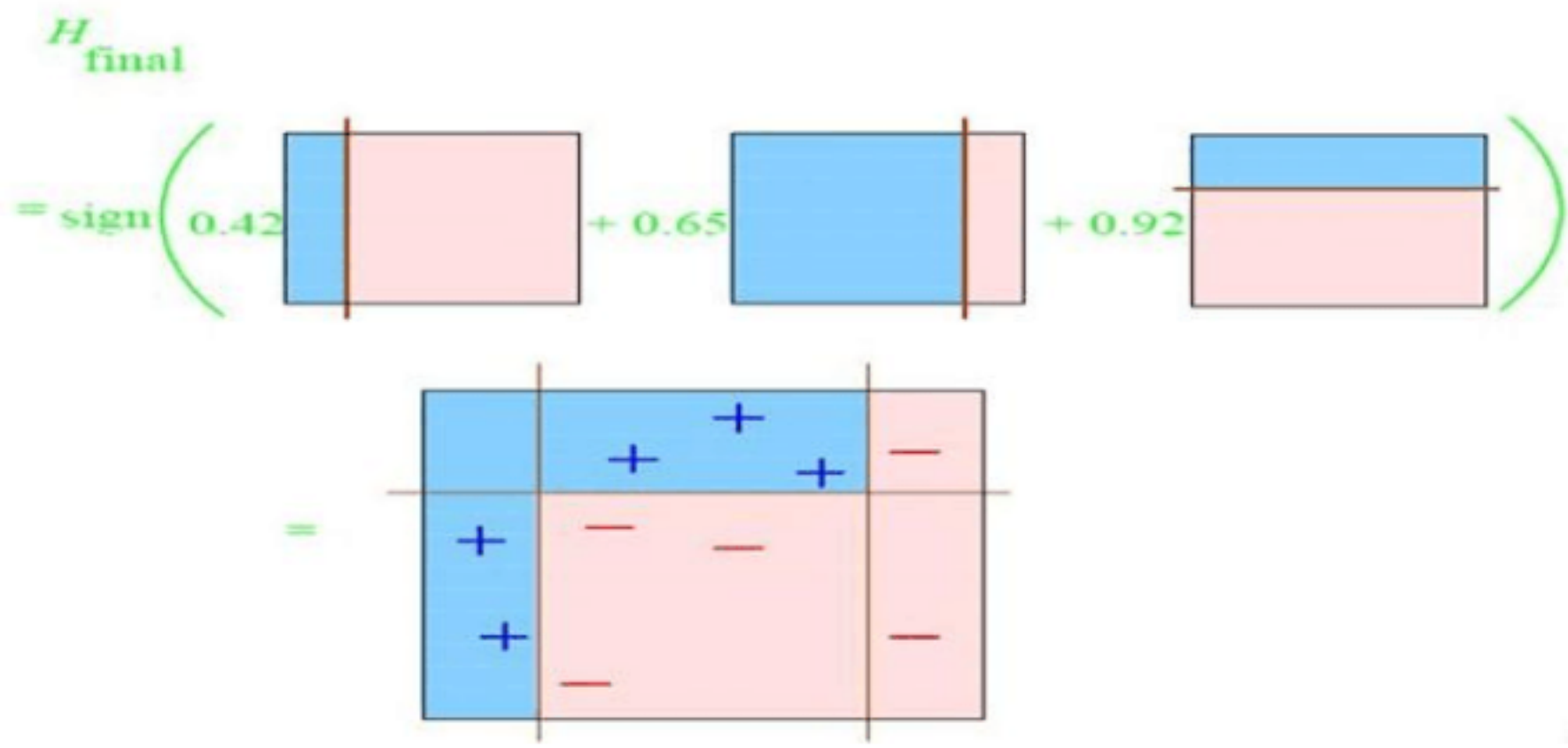
$\partial_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$ 可以算出来为 0.42. 然后就根据算法把分错的点权值变大。 如此迭代，最终完成 Adaboost 算法。



根据分类的正确率，得到一个新的样本分布 D3，一个子分类器 h2。



得到一个子分类器 h3
整合所有子分类器：



此例的阈值是 1, 即加权平均后对于蓝色中大于 1 的区域就划分为蓝色种类，其他部分为红色种类。 因此，从综合的效果来看， 即使是一个简单的分类器组合，也可以得到较好的分类结果。

3 本文的方法

Adaboost 算法的完整步骤如下：

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

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

如今， Adaboost 算法有了很大的发展，并且还出现了许多其他 boost 算法，比如 Gentleboost 和 Logitboost 算法等。从上图中我们可以看到 Adaboost 算法的一个详细过程。 Adaboost 是一种较有特点的算法，现总结如下：

1）每次迭代改变的是样本的分布，而不是重复采样。

2）样本分布的改变取决于样本是否被正确分类。
总是分类正确的样本权值低，而分类错误的样本权值高。

3）最终的结果是弱分类器的加权组合。
权值表示该弱分类器的性能。

下面我们看看 Adaboost 是如何进行训练的：

训练过程：

A. 初始化训练样本x_i权重$\mathcal{D}_1(i)\quad i=1,\ldots,N$

(1)若正负样本数目一致，则$\mathcal{D}_1(i)=\dfrac{1}{N}$

(2)若正负样本数目分别为N_+,N_-，则$\begin{cases}\text{正样本}\ \mathcal{D}_1(i)=\dfrac{1}{2N_+}\\ \text{负样本}\ \mathcal{D}_1(i)=\dfrac{1}{2N_-}\end{cases}$

B. *for* $m=1,\ldots,M$

(1)**训练弱分类器**$f_m(x)=L(\mathcal{D},\mathcal{D}_m)\in\{-1,+1\}$

(2)估计弱分类器$f_m(x)$的分类错误率e_m

如：$e_m=\dfrac{1}{2}\sum_{i=1}^{N}\mathcal{D}_m(i)\cdot\left|f_m(x_i)-y_i\right|$ [注：$e_m<0.5$]

B. *for* $m=1,\ldots,M$(续前)

(3)估计弱分类器$f_m(x)$的权重$c_m=\log\dfrac{1-e_m}{e_m}$

(4)基于弱分类器$f_m(x)$调整各样本权重，并归一化

调　整：$\mathcal{D}_{m+1}(i)=\mathcal{D}_m(i)\cdot\exp\left[c_m\cdot 1_{(f_m(x_i)\neq y_i)}\right]$

$$=\begin{cases}\mathcal{D}_m(i) & \text{若}f_m(x_i)=y_i\\ \mathcal{D}_m(i)\cdot\dfrac{1-e_m}{e_m} & \text{若}f_m(x_i)\neq y_i\end{cases}$$

归一化：$\mathcal{D}_{m+1}(i)\leftarrow\dfrac{\mathcal{D}_{m+1}(i)}{\sum_{j=1}^{N}\mathcal{D}_{m+1}(j)}\quad i=1,\ldots,N$

C. 强分类器

$$H(x)=\operatorname{sgn}\left[\sum_{m=1}^{M}c_m f_m(x)\right]$$

4 实验

（1）实验程序

这是用 Java 语言在 Eclipse 环境中写的一个小程序。下面的代码：

Adaboost 类的代码：

```
public class Adaboost {
    int [][] matrix = new int [8][8]; // 矩阵，行代表基分类器，列代表测试点
    Point[] point ;
    BaseClassifiers[] bc;

    Vector<Integer> h = new Vector<Integer>();
    Vector<Double> a = new Vector<Double>();

    // 预测点的 label
    public int judge(Point p) {
        double result = 0.0;
        for ( int i = 0; i < h.size(); i++) {
            double temp = a.get(i) * bc[ h.get(i)].classify(p);
            result += temp;
        }
        return result > 0 ? 1 : -1;
    }

    // 这个函数在对象生成后就应该被调用，但是参数由用户输入，所以没有把它放到生成函数中
    public void Adaboost( int T, double delt) { // 传过来 T=10 , delt=0.01 ;
        int len = this . point . length ; // 长度为 8，即 len=8
        double [] D = new double [len]; // 每个点的 权重
        // 初始时每个点权重应满足均匀分布
        for ( int i = 0; i < len; i++) {
            D[i] = 1.0 / len;
        }
        for ( int t = 1; t < T; t++) {
            double error = chooseHypothese(D);
            if (Math. abs (error - 0.5) < delt) {
                break ;
            }
            double at = 0.5 * Math. log ((1 - error) / error);
            a.add(at);
            double z = 2 * Math. sqrt (error * (1 - error));
            for ( int i = 0; i < len; i++) {
                D[i] = D[i] * Math. exp (-1 * at * point [i]. label
                                * matrix [ h.get( h.size() - 1)][i]) / z;
            }
        }
```

```
DecimalFormat df = new DecimalFormat( "0.00" );
for (Integer i : h) {
    System. out .print(i + "\t\t" );
}
System. out .println();
for (Double d : a) {
    String s = df.format(d);
    System. out .print(s + "\t\t" );
}
}

// 选择最小错误的分类器，返回错误值
public double chooseHypothese( double [] D) {
    int min = Integer. MAX_VALUE;
    double error = 1.1;
    for ( int i = 0; i < bc. length ; i++) {
        double temp = 0.0;
        for ( int j = 0; j < point . length ; j++) {
            if ( bc [i].classify( point [j]) != point [j]. label ) {
                temp += D[j];
            }
        }
        if (temp < error) {
            error = temp;
            min = i;
        }
    }
    h.add(min);
    return error;
}

public Adaboost(Point[] point, BaseClassifiers[] bc) {
    this . point = point;
    this . bc = bc;
    // 计算矩阵，行代表基分类器，列代表测试点，值为 1则表示分类器对
测试点分类为 1，-1 类似
    for ( int i = 0; i < bc. length ; i++) { // 遍历分类器
        for ( int j = 0; j < point. length ; j++) { // 遍历测试点
            matrix [i][j] = bc[i].classify(point[j]);
        }
    } }}
```

BaseClassifiers 类的代码：

```
public class BaseClassifiers {
```

```
char type ;     // 类型, x代表x=coordinate , y代表y=coordinate
int coordinate ;

public BaseClassifiers( char type, int coordinate){
    this . type =type;
    this . coordinate =coordinate;
}

// 在分类器右方或者上方返回 1, 否则返回 -1
public int classify(Point p){
    double x=p. x;
    double y=p. y;
    if ( this . type ='x' ){
        if ((x- this . coordinate >=0)){
            return 1;
        } else {
            return -1;
        }
    } else {
        if ((y- this . coordinate >=0)){
            return 1;
        } else {
            return -1;
        }
    }  }}
```

Point 类的代码：

```
public class Point {
    double x;
    double y;
    int label ;     // 标签0或1
    public Point( double x, double y, int value){
        this . x=x;
        this . y=y;
        this . label =value;
    }
    public String toString(){
        return x+" " +y+" " +label ;
    }
}
```

主程序 AdaboostTest 代码：

```
public class AdaboostTest {
```

```
@Test
public     void   test() {

      Point[] point =           new Point[8];
      point[0] =        new Point(0.5, 0.5, -1);
      point[1] =        new Point(1.5, 0.5, -1);
      point[2] =        new Point(4.5, 0.5, 1);
      point[3] =        new Point(0.5, 1.5, -1);
      point[4] =        new Point(2.5, 2.5, -1);
      point[5] =        new Point(2.5, 0.5, 1);
      point[6] =        new Point(4.5, 3.5, 1);
      point[7] =        new Point(2.5, 5.5, 1);

      //    弱分类器为 x=1,x=2,x=3,x=4,y=1,y=2,y=3,y=4
      BaseClassifiers[] bc =            new BaseClassifiers[8];
      bc[0] =      new BaseClassifiers(        'x'   , 1);
      bc[1] =      new BaseClassifiers(        'x'   , 2);
      bc[2] =      new BaseClassifiers(        'x'   , 3);
      bc[3] =      new BaseClassifiers(        'x'   , 4);
      bc[4] =      new BaseClassifiers(        'y'   , 1);
      bc[5] =      new BaseClassifiers(        'y'   , 2);
      bc[6] =      new BaseClassifiers(        'y'   , 3);
      bc[7] =      new BaseClassifiers(        'y'   , 4);

      Adaboost ada =      new Adaboost(point, bc);
            ada.Adaboost(10, 0.01);
  }}
```

（2）实验结果

```
1              2              1              6              1              6
 0.97           0.90           0.44           0.67           0.39           0.28

1              2              1
 0.22           0.19           0.16
```

（3）结果分析

从上面结果可知，在 x 轴等于 1，y 轴从 2 到 6 再至 1 的过程，其下面对应的错误率一直减小， 直至在可接受的范围内为止， 这样就正确的将输入的数据分类。

5 讨论与结束语

对于 Adaboost 的讨论，目前比较热门的就是人脸识别了。尤其是当今互联网、电子设备的大量普及， 我们可以从网络中大量的图片或录像中识别出有人脸的图像，精度高的话可以筛选出某个人的头像； 不管是照相机还是手机上的照相机也会拥有人脸识别，提高照相的效果。像这样的应用还有很多很多，当然，追

求质量和速度的互联网时代， Adaboost 的改进是一门热门的课题。

Adaboost 分类算法具有非常高的精度，我们可以采用多种方法构建子分类器，然后用 Adaboost 这种框架进行综合分类（即大家所说的三个臭皮匠赛过一个诸葛亮）。另外，Adaboost 易于执行，系统检测运行的速度很高， 但是 Adaboost 算法本身由于要进行多次训练， 当次数 T 过大时就比较耗时， 因此使整个系统的训练时间加长。也正是由于 Adaboost 算法训练时间过长，从而大大地降低了该算法的应用范围。再者， Adaboost 算法框架是建立在弱分类器的选择上，如果弱分类器选择不恰当， 也会给此次执行效果大打折扣， 过度依赖弱分类器也是其一个缺点。当然，容易受到噪声干扰也是大部分算法的不足之处。所以，在用 Adaboost 进行分析时，需要考虑到这几点。

火龙果 • 整理
uml.org.cn

参考文献：

[1] 左登宇 .基于 Adaboost 算法检测研究 [J]200937-45

[2] 柴梅平 ,朱明 .基于彩色分割的人脸检测算法的研究闭 ,计算机测量与控制 ,2006,14(l):1113.

[3] VIOLAP.RapidobjectdetectionusingaBoostedcascadeofsim-plefeatur[J].PrcIEEEConfernceonComputerVisionandPaternRecogition,2001.511 一 518.

[4] FreundY.BoostingaWeakLearningAlgorithmBymajor-ity[J].InformationandComputation,1995 , 121(2)：256285.

[5] 黄文杰 ,陈斌 .一种快速图像处理的积分图方法 [J]. 计算机应用 2005,25(Z1),266 一 268.

[6] 陈茂林 ,戚飞虎 .自组织隐马尔可夫模型的人脸检测研究 [J]. 计算机学报 ,2002,25(II):11651169.

[7] 孔凡之 ,张兴周 ,谢耀菊 .基于 Adaboost 的人脸检测技术 [J]. 应用科学 ,2O05,32(6):710

[8] 金王倩 ,陈斌 ,黄文杰 .Adaboost 算法在喷码图像识别的应用 [J] 计算机应用 2006.26(9).

[9] T. Kohonen,Self-Organization and Associative Memory,Springer 1989.

[10] K. K. Sung,T. Poggio 、 Example-Based Learning for View-Based Human Face Detection 、 Technical Report AI Memo 1521 , Massachusetts Inst. of Technology AI Lab, 1994.

[11] Freund Y, Schapire R. E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences, 1997, 55(1):119-139.

[12] Kearns M.The Computational Complexity of Machine Learning 、 Cambridge: MIT Press, 1990.

[13] Viola P.Jones M. J.Robust Real-time Object Detection 、 Cambridge Research Laboratory, Technical Report Series、 CRL 2001/01.

[14] 宋春雷 ,王龙等 .学习理论与鲁棒控制理论与应用 .Vol 17(5), 633-636,2000/10.

[15] 李军 ,董元方 .Boosting 算法的理论分析 .长春工业大学学报 [J],2008,10(5):29.