



山东大学
SHANDONG UNIVERSITY

比较支持向量机、AdaBoost、 逻辑斯谛回归模型的学习策略 与算法报告 (2018 届)

学院： 控制科学与工程学院

专业： 自动化

姓名： 范钧宇

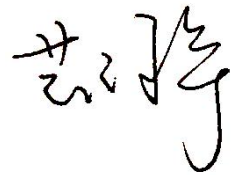
学号： 201800171021

完成时间： 2021 年 5 月 7 日

原创性声明

兹提交的实验报告，是本人在指导老师指导下独立完成的成果。本人在设计中参考的其他个人或集体的成果，均在设计作品文字说明中以明确方式标明。本人依法享有和承担由此设计作品而产生的权利和责任。

声明人（签名）：

A handwritten signature in black ink, appearing to be '赵玲' (Zhao Ling), written in a cursive style.

2021 年 5 月 7 日

目录

一、支持向量机.....	2
1. SVM 简介.....	2
2. 线性 SVM 算法原理.....	2
3. 非线性 SVM 算法原理.....	9
二、Adaboost 算法.....	10
1. 什么是 AdaBoost 算法?	10
2. Adaboost 的 7 个优缺点.....	11
3. 回顾 boosting 算法的基本原理.....	12
三、逻辑斯蒂回归.....	19
1. 引言.....	19
2. logistic distribution (逻辑斯蒂分布)	21
3. 从逻辑斯蒂分布 到 逻辑斯蒂回归模型.....	22
4. 带惩罚项的 LR 回归.....	24
5. 为什么逻辑回归比线性回归要好?	24

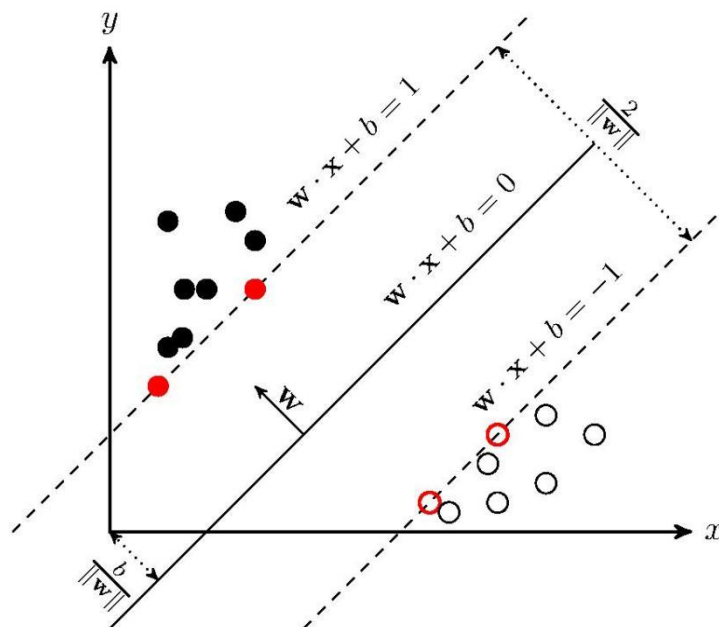
一、支持向量机

1.SVM 简介

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 还包括核技巧，这使它成为实质上的非线性分类器。SVM 的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的学习算法就是求解凸二次规划的最优化算法。

2.线性 SVM 算法原理

SVM 学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。如下图所示， $\omega \cdot x + b = 0$ 即为分离超平面，对于线性可分的数据集来说，这样的超平面有无穷多个（即感知机），但是几何间隔最大的分离超平面却是唯一的。



在推导之前，先给出一些定义。假设给定一个特征空间上的训练数据集

$$T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$$

其中， $\mathbf{x}_i \in \mathbb{R}^n$ ， $y_i \in \{+1, -1\}$ ， $i = 1, 2, \dots, N$ ， \mathbf{x}_i 为第*i*个特征向量， y_i 为

类标记，当它等于+1 时为正例；为-1 时为负例。再假设训练数据集是线性可分的。

几何间隔：对于给定的数据集 T 和超平面 $\omega \cdot x + b = 0$ ，定义超平面关于样本点 (x_i, y_i) 的几何间隔为

$$\gamma_i = y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}_i + \frac{b}{\|\mathbf{w}\|} \right)$$

超平面关于所有样本点的几何间隔的最小值为

$$\gamma = \min_{i=1,2,\dots,N} \gamma_i$$

实际上这个距离就是我们所谓的支持向量到超平面的距离。

根据以上定义，SVM 模型的求解最大分割超平面问题可以表示为以下约束最优化问题

$$\max_{\mathbf{w}, b} \gamma$$

$$s.t. \quad y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}_i + \frac{b}{\|\mathbf{w}\|} \right) \geq \gamma, i = 1, 2, \dots, N$$

将约束条件两边同时除以 γ ，得到

$$y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|\gamma} \cdot \mathbf{x}_i + \frac{b}{\|\mathbf{w}\|\gamma} \right) \geq 1$$

因为 $\|\omega\|$ ， γ 都是标量，所以为了表达式简洁起见，令

$$\mathbf{w} = \frac{\mathbf{w}}{\|\mathbf{w}\|\gamma}$$

$$b = \frac{b}{\|\mathbf{w}\|\gamma}$$

得到

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, i = 1, 2, \dots, N$$

又因为最大化 γ ，等价于最大化 $\frac{1}{\|\omega\|}$ ，也就等价于最小化 $\frac{1}{2} \|\omega\|^2$ ($\frac{1}{2}$ 是为了后面求导

以后形式简洁，不影响结果），因此 SVM 模型的求解最大分割超平面问题又可以表示为以下约束最优化问题

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

$$s.t. \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, N$$

这是一个含有不等式约束的凸二次规划问题，可以对其使用拉格朗日乘子法得到其对偶问题（dual problem）。

首先，我们将有约束的原始目标函数转换为无约束的新构造的拉格朗日目标函数

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1)$$

其中 α_i 为拉格朗日乘子，且 $\alpha_i \geq 0$ 。现在我们令

$$\theta(\mathbf{w}) = \max_{\alpha_i \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha})$$

当样本点不满足约束条件时，即在可行解区域外：

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) < 1$$

此时，将 α_i 设置为无穷大，则 $\theta(\omega)$ 也为无穷大。

当样本点满足约束条件时，即在可行解区域内：

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

此时， $\theta(\omega)$ 为原函数本身。于是，将两种情况合并起来就可以得到我们新的目标函数

$$\theta(\mathbf{w}) = \begin{cases} \frac{1}{2} \|\mathbf{w}\|^2, & \mathbf{x} \in \text{可行区域} \\ +\infty, & \mathbf{x} \in \text{不可行区域} \end{cases}$$

于是原约束问题就等价于

$$\min_{\mathbf{w}, b} \theta(\mathbf{w}) = \min_{\mathbf{w}, b} \max_{\alpha_i \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha}) = p^*$$

看一下我们的新目标函数，先求最大值，再求最小值。这样的话，我们首先就要面对带有需要求解的参数 ω 和 b 的方程，而 α_i 又是不等式约束，这个求解过程不好做。所以，我们需要使用拉格朗日函数对偶性，将最小和最大的位置交换一下，这样就变成了：

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w}, b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = d^*$$

要有 $p^* = d^*$ ，需要满足两个条件：

① 优化问题是凸优化问题

② 满足 KKT 条件

首先，本优化问题显然是一个凸优化问题，所以条件一满足，而为了满足条件二，即要求

$$\begin{cases} \alpha_i \geq 0 \\ y_i (\mathbf{w}_i \cdot \mathbf{x}_i + b) - 1 \geq 0 \\ \alpha_i (y_i (\mathbf{w}_i \cdot \mathbf{x}_i + b) - 1) = 0 \end{cases}$$

为了得到求解对偶问题的具体形式，令 $L(\omega, b, \alpha)$ 对 ω 和 b 的偏导为 0，可得

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

将以上两个等式带入拉格朗日目标函数，消去 ω 和 b ，得

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) - \sum_{i=1}^N \alpha_i y_i \left(\left(\sum_{j=1}^N \alpha_j y_j \mathbf{x}_j \right) \cdot \mathbf{x}_i + b \right) + \sum_{i=1}^N \alpha_i \\ &= -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^N \alpha_i \end{aligned}$$

即

$$\min_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^N \alpha_i$$

求 $\min_{\omega, b} L(\omega, b, \alpha)$ 对 α 的极大，即是对偶问题

$$\max_{\alpha} -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^N \alpha_i$$

$$s. t. \quad \sum_{i=1}^N \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 1, 2, \dots, N$$

把目标式子加一个负号，将求解极大转换为求解极小

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) - \sum_{i=1}^N \alpha_i$$

$$s. t. \quad \sum_{i=1}^N \alpha_i y_i = 0$$

$$\alpha_i \geq 0, i = 1, 2, \dots, N$$

现在我们的优化问题变成了如上的形式。对于这个问题，我们有更高效的优化算法，即序列最小优化（SMO）算法。这里暂时不展开关于使用 SMO 算法求解以上优化问题的细节，下一篇文章再加以详细推导。

我们通过这个优化算法能得到 α^* ，再根据 α^* ，我们就可以求解出 ω 和 b ，进而求得我们最初的目的：找到超平面，即”决策平面”。

前面的推导都是假设满足 KKT 条件下成立的，KKT 条件如下

$$\begin{cases} \alpha_i \geq 0 \\ y_i (\mathbf{w}_i \cdot \mathbf{x}_i + b) - 1 \geq 0 \\ \alpha_i (y_i (\mathbf{w}_i \cdot \mathbf{x}_i + b) - 1) = 0 \end{cases}$$

另外，根据前面的推导，还有下面两个式子成立

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

由此可知在 α^* 中，至少存在一个 $\alpha^* > 0$ （反证法可以证明，若全为0，则 $\omega=0$ ，矛盾），对此 j 有

$$y_j (\mathbf{w}^* \cdot \mathbf{x}_j + b^*) - 1 = 0$$

因此可以得到

$$\mathbf{w}^* = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i$$

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}_j)$$

对于任意训练样本 (x_i, y_i) ，总有 $\alpha_i = 0$ 或者 $y_i(\omega \cdot x_j + b) = 1$ 。若 $\alpha_i = 0$ 则该样本不会在最后求解模型参数的式子中出现。若 $\alpha_i > 0$ ，则必有 $y_i(\omega \cdot x_j + b) = 1$ ，所对应的样本点位于最大间隔边界上，是一个支持向量。这显示出支持向量机的一个重要性质：训练完成后，大部分的训练样本都不需要保留，最终模型仅与支持向量有关。

到这里都是基于训练集数据线性可分的假设下进行的，但是实际情况几乎不存在完全线性可分的数据，为了解决这个问题，引入了“软间隔”的概念，即允许某些点不满足约束。

$$y_j (\mathbf{w} \cdot \mathbf{x}_j + b) \geq 1$$

采用 hinge 损失，将原优化问题改写为

$$\min_{\mathbf{w}, b, \xi_i} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i$$

$$s.t. \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, \quad i = 1, 2, \dots, N$$

其中 ξ_i 为“松弛变量”， $\xi_i = \max(0, 1 - y_i(\omega \cdot x_i + b))$ ，即一个 hinge 损失函数。

每一个样本都有一个对应的松弛变量，表征该样本不满足约束的程度。 $C > 0$ 称为惩罚参数， C 值越大，对分类的惩罚越大。跟线性可分求解的思路一致，同样这里先用拉格朗日乘子法得到拉格朗日函数，再求其对偶问题。

综合以上讨论，我们可以得到线性支持向量机学习算法如下：

输入：训练数据集 $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$ 其中, $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \{+1, -1\}, i = 1, 2, \dots, N$;

输出：分离超平面和分类决策函数

(1) 选择惩罚参数 $C > 0$ ，构造并求解凸二次规划问题

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) - \sum_{i=1}^N \alpha_i$$

$$s. t. \quad \sum_{i=1}^N \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, i = 1, 2, \dots, N$$

得到最优解 $\boldsymbol{\alpha}^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$

(2) 计算

$$\mathbf{w}^* = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i$$

选择 $\boldsymbol{\alpha}^*$ 的一个分量 α_i^* 满足条件 $0 < \alpha_i^* < C$ ，计算

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}_j)$$

(3) 求分离超平面

$$\mathbf{w}^* \cdot \mathbf{x} + b^* = 0$$

分类决策函数：

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^* \cdot \mathbf{x} + b^*)$$

3.非线性 SVM 算法原理

对于输入空间中的非线性分类问题,可以通过非线性变换将它转化为某个维特征空间中的线性分类问题,在高维特征空间中学习线性支持向量机。由于在线性支持向量机学习的对偶问题里,目标函数和分类决策函数都只涉及实例和实例之间的内积,所以不需要显式地指定非线性变换,而是用核函数替换当中的内积。核函数表示,通过一个非线性转换后的两个实例间的内积。具体地, $K(\mathbf{x}, \mathbf{z})$ 是一个函数,或正定核,意味着存在一个从输入空间到特征空间的映射 $\phi(\mathbf{x})$, 对任意输入空间中的 \mathbf{x}, \mathbf{z} , 有

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$$

在线性支持向量机学习的对偶问题中,用核函数 $K(\mathbf{x}, \mathbf{z})$ 替代内积,求解得到的就是非线性支持向量机

$$f(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^N \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^* \right)$$

综合以上讨论,我们可以得到非线性支持向量机学习算法如下:

输入: 训练数据集 $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$ 其中, $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \{+1, -1\}, i = 1, 2, \dots, N$;

输出: 分离超平面和分类决策函数

(1) 选取适当的核函数 $K(\mathbf{x}, \mathbf{z})$ 和惩罚参数 $C > 0$, 构造并求解凸二次规划问题

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^N \alpha_i$$

$$s.t. \quad \sum_{i=1}^N \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, i = 1, 2, \dots, N$$

得到最优解 $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$

(2) 计算

选择 α^* 的一个分量 α_j^* 满足条件 $0 < \alpha_j^* < C$, 计算

$$b^* = y_j - \sum_{i=1}^N \alpha_i^* y_i K(x_i, x_j)$$

(3) 分类决策函数:

$$f(x) = \text{sign} \left(\sum_{i=1}^N \alpha_i^* y_i K(x, x_i) + b^* \right)$$

介绍一个常用的核函数——高斯核函数

$$K(x, z) = \exp \left(-\frac{\|x - z\|^2}{2\sigma^2} \right)$$

对应的 SVM 是高斯径向基函数分类器, 在此情况下, 分类决策函数为

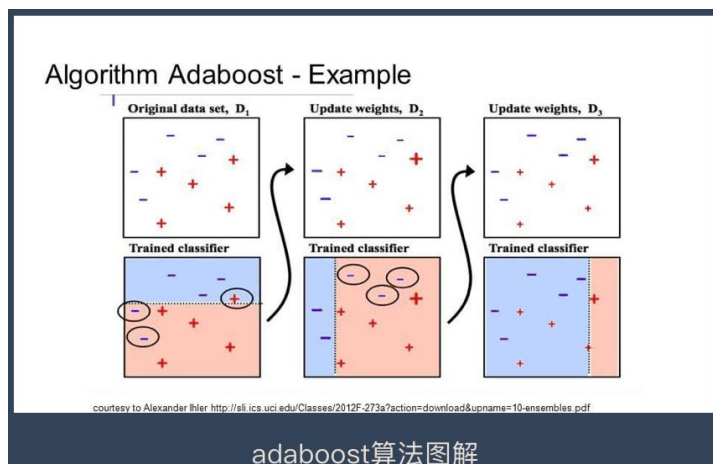
$$f(x) = \text{sign} \left(\sum_{i=1}^N \alpha_i^* y_i \exp \left(-\frac{\|x - z\|^2}{2\sigma^2} \right) + b^* \right)$$

二、Adaboost 算法

1. 什么是 AdaBoost 算法?

Boosting 是一种集合技术, 试图从许多弱分类器中创建一个强分类器。这是通过从训练数据构建模型, 然后创建第二个模型来尝试从第一个模型中纠正错误来完成的。添加模型直到完美预测训练集或添加最大数量的模型。

AdaBoost 是第一个为二进制分类开发的真正成功的增强算法。这是理解助力的最佳起点。现代助推方法建立在 AdaBoost 上, 最著名的是随机梯度增强机。



AdaBoost 用于短决策树。在创建第一个树之后，每个训练实例上的树的性能用于加权创建的下一个树应该关注每个训练实例的注意力。难以预测的训练数据被赋予更多权重，而易于预测的实例被赋予更少的权重。模型一个接一个地顺序创建，每个模型更新训练实例上的权重，这些权重影响序列中下一个树所执行的学习。构建完所有树之后，将对新数据进行预测，并根据训练数据的准确性对每棵树的性能进行加权。

因为通过算法如此关注纠正错误，所以必须删除带有异常值的干净数据。

2. Adaboost 的 7 个优缺点

Adaboost优点	Adaboost优点
很好的利用了弱分类器进行级联	AdaBoost弱分类器数目不太好设定
可以将不同的分类算法作为弱分类器	数据不平衡导致分类精度下降
AdaBoost具有很高的精度	训练比较耗时
AdaBoost充分考虑的每个分类器的权重	

(1) AdaBoost 算法优点：

很好的利用了弱分类器进行级联；

可以将不同的分类算法作为弱分类器；

AdaBoost 具有很高的精度；

相对于 bagging 算法和 Random Forest 算法，AdaBoost 充分考虑的每个分类器的权重；

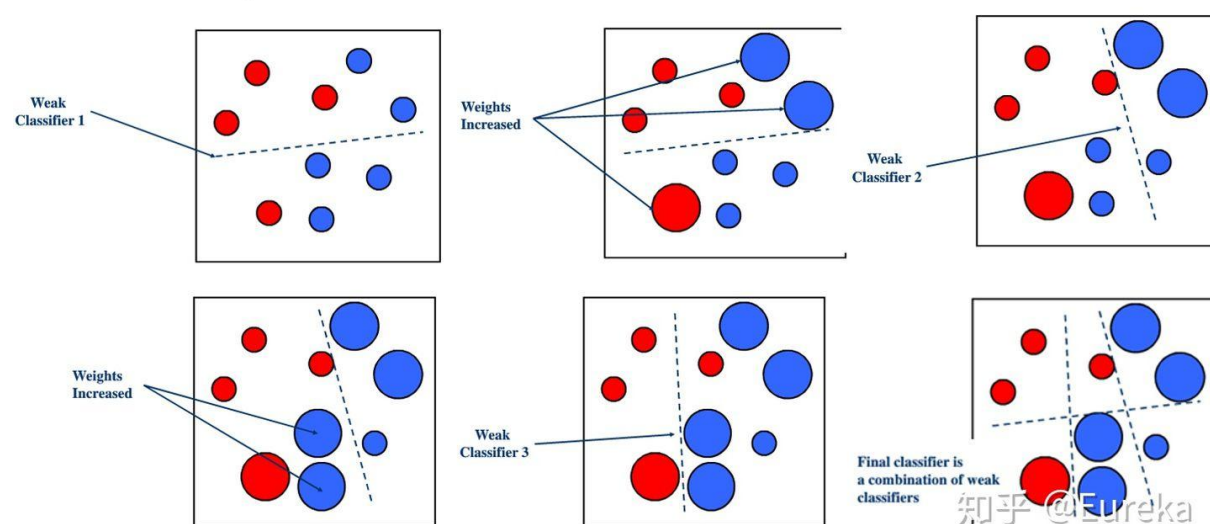
(2) Adaboost 算法缺点：

AdaBoost 迭代次数也就是弱分类器数目不太好设定，可以使用交叉验证来进行确定；

数据不平衡导致分类精度下降；

训练比较耗时，每次重新选择当前分类器最好切分点；

3.回顾 boosting 算法的基本原理



Boosting 算法的工作机制是首先从训练集用初始权重训练出一个弱学习器 1，根据弱学习的学习误差率表现来更新训练样本的权重，使得之前弱学习器 1 学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器 2 中得到更多的重视。然后基于调整权重后的训练集来训练弱学习器 2.，如此重复进行，直到弱学习器数达到事先指定的数目 T ，最终将这 T 个弱学习器通过集合策略进行整合，得到最终的强学习器。

不过有几个具体的问题 Boosting 算法没有详细说明。

- (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) 使用何种结合策略?

4.提升方法 AdaBoost 算法

(1) 提升方法的基本思路

强可学习(strongly learnable)和弱可学习(weakly learnable)

1) 在概率近似正确(probably approximately correct, PAC)学习的框架中, 一个概念(类), 如果存在一个多项式的学习算法能够学习它, 并且正确率很高, 称这个概念是强可学习的;

2) 一个概念(类), 如果存在一个多项式的学习算法能够学习它, 学习的正确率仅比随机猜测略好, 则称这个概念是弱可学习的。

3) 在 PAC 学习的框架下, 一个概念是强可学习的充分必要条件是这个概念是弱可学习。

$$\left. \begin{array}{l} h_1(x) \in \{-1, +1\} \\ h_2(x) \in \{-1, +1\} \\ \vdots \\ h_T(x) \in \{-1, +1\} \end{array} \right\} \quad H_T(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

Weak classifiers strong classifier

slightly better than random 知乎 @Eureka

4) 怎样获得不同的弱分类器?

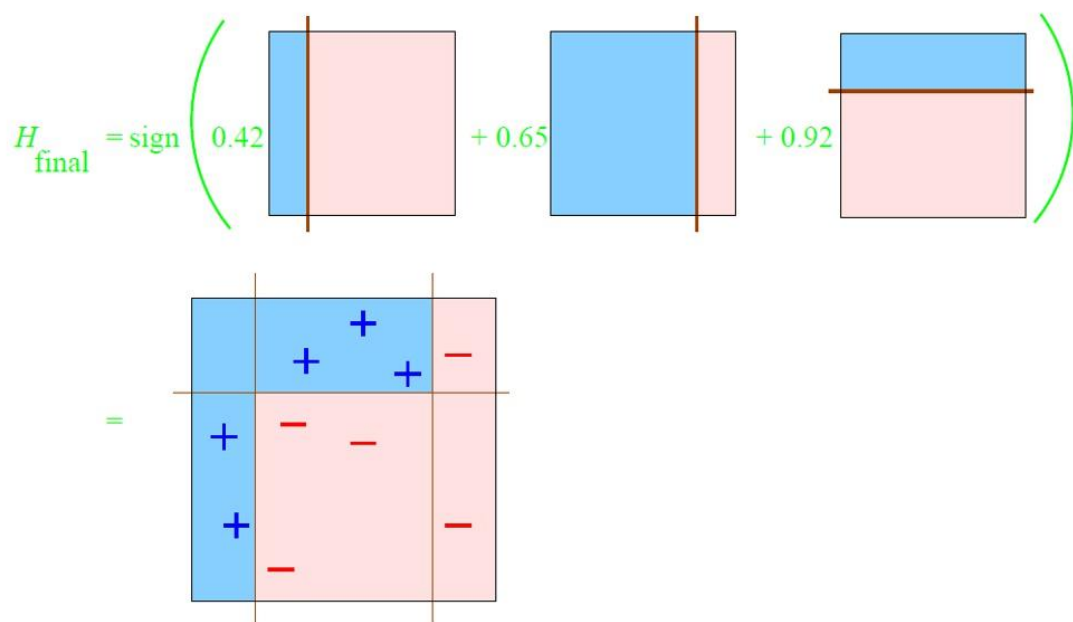
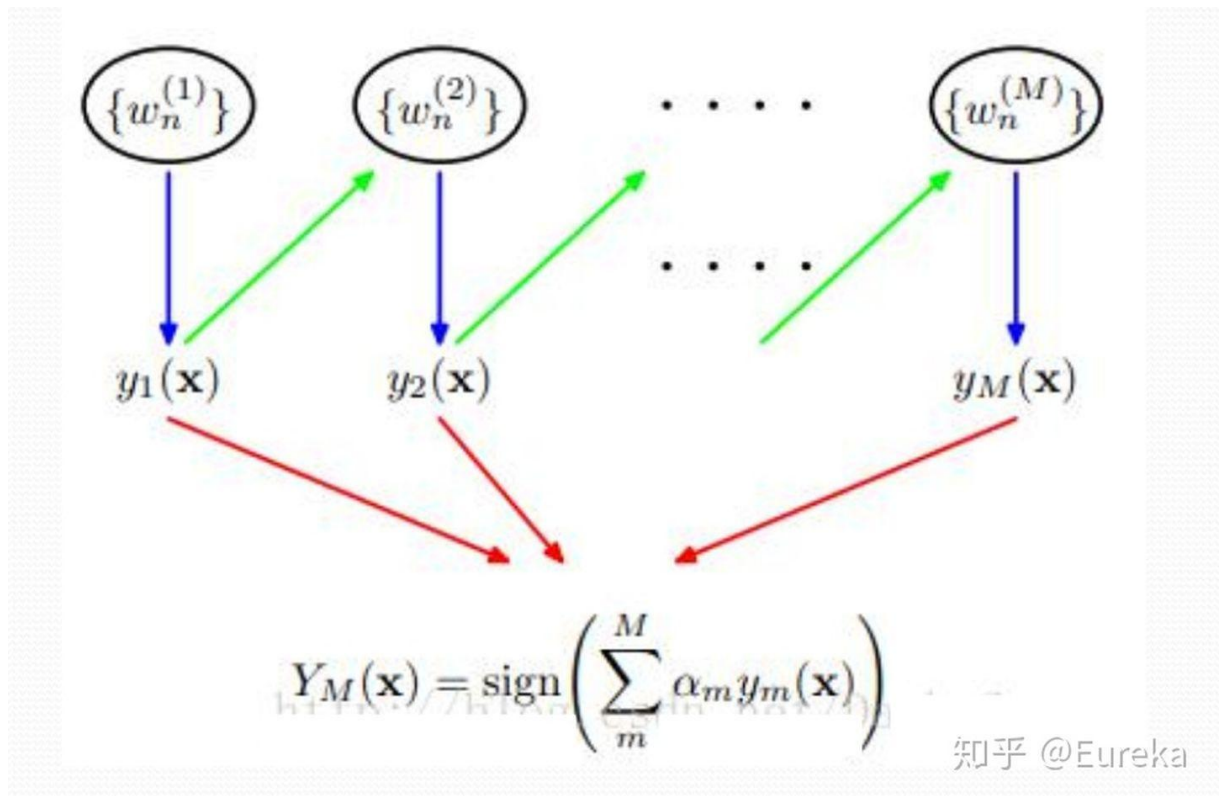
- ①使用不同的弱学习算法得到不同基本学习器
- ②使用相同的弱学习算法, 但用不同的参数
- ③相同输入对象的不同表示凸显事物不同的特征
- ④使用不同的训练集: 装袋(bagging)也称为自举汇聚法(bootstrap aggregating)与提升(boosting)

5) 怎样组合弱分类器?

- ①多专家组合: 一种并行结构, 所有的弱分类器都给出各自的预测结果, 通过“组合器”把这些预测结果转换为最终结果。eg. 投票(voting)及其变种、混合专家模型

②多级组合：一种串行结构，其中下一个分类器只在前一个分类器预测不够准（不够自信）的实例上进行训练或检测。eg. 级联算法（cascading）

（2）AdaBoost 算法（分类）



1) AdaBoost 算法:

输入: 训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, 其中 $x_i \in \mathcal{X} \subseteq R^n$
 $y_i \in \mathcal{Y} = \{+1, -1\}, i = 1, 2, \dots, N$; 弱学习算法输出: 分类器 $G(x)$

①初始化训练数据的权值分布

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

②对 $m=1, 2, \dots, M$

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

$$G_m(x) : \mathcal{X} \rightarrow \{-1, +1\}$$

计算 $G_m(x)$ 在训练数据集上的分类误差率

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

计算 $G_m(x)$ 的系数

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

更新训练数据集的权值分布

$$\begin{aligned} D_{m+1} &= (w_{m+1,1}, \dots, w_{m+1,i}, \dots, w_{m+1,N}) \\ w_{m+1,i} &= \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \\ &= \begin{cases} \frac{w_{mi}}{Z_m} \exp(-\alpha_m), & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} \exp(\alpha_m), & G_m(x_i) \neq y_i \end{cases} \quad i = 1, 2, \dots, N \end{aligned}$$

其中, Z_m 是规范化因子

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

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

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

注：对于 Adaboost 多元分类算法，其实原理和二元分类类似，最主要区别在弱分类器的系数上。比如 Adaboost SAMME 算法，它的弱分类器的系数：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} + \log(R - 1)$$

其中 R 为类别数。从上式可以看出，如果是二元分类， $R=2$ ，则上式和我们的二元分类算法中的弱分类器的系数一致。

2) 算法说明：

①计算 $G_m(x)$ 在训练数据集上的分类误差率

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

这里 $\sum_{i=1}^N w_{mi} = 1$ ，这表明 $G_m(x)$ 在加权的训练数据集上的分类错误率是被 $G_m(x)$ 分类样本的权值之和，由此可以看出数据全值分布 D_m 与基本分类器 $G_m(x)$ 的分类误差率的关系。

②当 $e_m \leq \frac{1}{2}$ 时， $\alpha \geq 0$ ，并且 α 随着 e_m 的减少而增大，所以误分类误差率越小的基本分类器在最终分类器的作用越大。

③更新训练数据的权值分布为下一轮做准备，式可以写成：

$$w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases}$$

由此可知，被基本分类器 $G_m(x)$ 误分类样本的权值得以扩大，而被正确分类的样本的权值得以缩小。

④这里系数 α_m 表示了基本分类器 G_m 的重要性，所以 α_m 之和并不为 1。 $f(x)$ 的符号决定实例 x 类， $f(x)$ 的绝对值表示分类的确信度。

(3) AdaBoost 算法（回归）

上面是 Adaboost 做分类，下面来看看 Adaboost 做回归时误差率、权重系数如何选择：

先看看回归问题的误差率的问题，对于第 m 个弱学习器，计算他在训练集上的最大误差：

$$E_m = \max |y_i - G_m(x_i)| \quad i = 1, 2, \dots, N$$

然后计算每个样本的相对误差：

$$e_{mi} = \frac{|y_i - G_m(x_i)|}{E_m}$$

这里是误差损失为线性时的情况，如果我们用平方误差，则

$$e_{mi} = \frac{(y_i - G_m(x_i))^2}{E_m^2}$$

如果我们用的是指数误差，则

$$e_{mi} = 1 - \exp\left(\frac{-y_i + G_m(x_i)}{E_m}\right)$$

最终得到第 m 个弱学习器的误差率：

$$e_m = \sum_{i=1}^N w_{mi} e_{mi}$$

我们再来看看如何得到弱学习器权重系数 α 。这里有：

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

对于更新更新样本权重 D ，第 $m+1$ 个弱学习器的样本集权重系数为：

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \alpha_m^{1-e_{mi}}$$

这里 Z_m 是规范化因子：

$$Z_m = \sum_{i=1}^N w_{mi} \alpha_m^{1-e_{mi}}$$

最后是结合策略，和分类问题稍有不同，采用的是对加权的弱学习器取中位数的方法，最终的强回归器为：

$$f(x) = \sum_{i=1}^N \left(\ln \frac{1}{\alpha_m}\right) g(x)$$

1) AdaBoost 算法：

输入：训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ，其中 $x_i \in \mathcal{X} \subseteq R^n$
 $y_i \in \mathcal{Y} = \{+1, -1\}, i = 1, 2, \dots, N$ ；弱学习算法输出：分类器 $G(x)$

①初始化训练数据的权值分布

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

②对 $m=1, 2, \dots, M$

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

$$G_m(x) : \mathcal{X} \rightarrow \{-1, +1\}$$

计算 $G_m(x)$ 在训练数据集上的分类误差率

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

计算 $G_m(x)$ 的系数

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

更新训练数据集的权值分布

$$\begin{aligned} D_{m+1} &= (w_{m+1,1}, \dots, w_{m+1,i}, \dots, w_{m+1,N}) \\ w_{m+1,i} &= \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \\ &= \begin{cases} \frac{w_{mi}}{Z_m} \exp(-\alpha_m), & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} \exp(\alpha_m), & G_m(x_i) \neq y_i \end{cases} \quad i = 1, 2, \dots, N \end{aligned}$$

其中， Z_m 是规范化因子

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

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

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

注：对于 Adaboost 多元分类算法，其实原理和二元分类类似，最主要区别在弱分类器的系数上。比如 Adaboost SAMME 算法，它的弱分类器的系数：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} + \log(R - 1)$$

其中 R 为类别数。从上式可以看出，如果是二元分类， $R=2$ ，则上式和我们的二元分类算法中的弱分类器的系数一致。

2) 算法说明：

①计算 $G_m(x)$ 在训练数据集上的分类误差率

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

这里 $\sum_{i=1}^N w_{mi} = 1$ ，这表明 $G_m(x)$ 在加权的训练数据集上的分类错误率是被 $G_m(x)$ 分类样本的权值之和，由此可以看出数据全值分布 D_m 与基本分类器 $G_m(x)$ 的分类误差率的关系。

②当 $e_m \leq \frac{1}{2}$ 时， $\alpha \geq 0$ ，并且 α 随着 e_m 的减少而增大，所以误分类误差率越小的基本分类器在最终分类器的作用越大。

③更新训练数据的权值分布为下一轮做准备，式可以写成：

$$w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases}$$

由此可知，被基本分类器 $G_m(x)$ 误分类样本的权值得以扩大，而被正确分类的样本的权值得以缩小。

④这里系数 α_m 表示了基本分类器 G_m 的重要性，所以 α_m 之和并不为 1。 $f(x)$ 的符号决定实例 x 类， $f(x)$ 的绝对值表示分类的确信度。

三、逻辑斯蒂回归

1.引言

LR 回归，虽然这个算法从名字上来看，是回归算法，但实际上是一个分类算

法，学术界也叫它 logit regression, maximum-entropy classification (MaxEnt)或者是 the log-linear classifier。在机器学习算法中，有几十种分类器，LR 回归是最常用的一个。

logit 和 logistic 模型的区别：

二者的根本区别在于广义化线性模型中的联系函数的形式。logit 采用对数形式 $\log(p)$ ，logistic 形式为 $\log(p/(1-p))$ 。

当因变量是多类的，可以采用 logistic，也可以用 logit，计算结果并无多少差别。

LR 回归是在线性回归模型的基础上，使用 *sigmoid* 函数，将线性模型 $w^T x$ 的结果压缩到 $[0, 1]$ 之间，使其拥有概率意义。其本质仍然是一个线性模型，实现相对简单。在广告计算和推荐系统中使用频率极高，是 CTR 预估模型的基本算法。同时，LR 模型也是深度学习的基本组成单元。

LR 回归属于概率性判别式模型，之所谓是概率性模型，是因为 LR 模型是有概率意义的；之所以是判别式模型，是因为 LR 回归并没有对数据的分布进行建模，也就是说，LR 模型并不知道数据的具体分布，而是直接将判别函数，或者说是分类超平面求解了出来。

一般来说，分类算法都是求解 $p(C_k|x)$ ，即对于一个新的样本，计算其条件概率 $p(C_k|x)$ 。这个可以看做是一个后验概率，其计算可以基于贝叶斯公式得到：
$$p(C_k|x) = p(x|C_k)p(C_k) / \sum_{k=1}^K p(x|C_k)p(C_k)$$
，其中 $p(x|C_k)$ 是类条件概率密度， $p(C_k)$ 是类的概率先验。使用这种方法的模型，称为是生成模型，即： $p(C_k|x)$ 是由 $p(x|C_k)$ 和 $p(C_k)$ 生成的。

分类算法所得到的 $p(C_k|x)$ 可以将输入空间划分成许多不相交的区域，这些区域之间的分隔面被称为判别函数（也称为分类面），有了判别函数，就可以进行分类了，上面生成模型，最终也是为了得到判别函数。如果直接对判别函数进行求解，得到判别面，这种方法，就称为判别式法。LR 就属于这种方法。

2.logistic distribution （逻辑斯蒂分布）

之前说过，LR 回归是在线性回归模型的基础上，使用 sigmoid 函数得到的。关于线性模型，在前面的文章中已经说了很多了。这里就先介绍一下， sigmoid 函数。

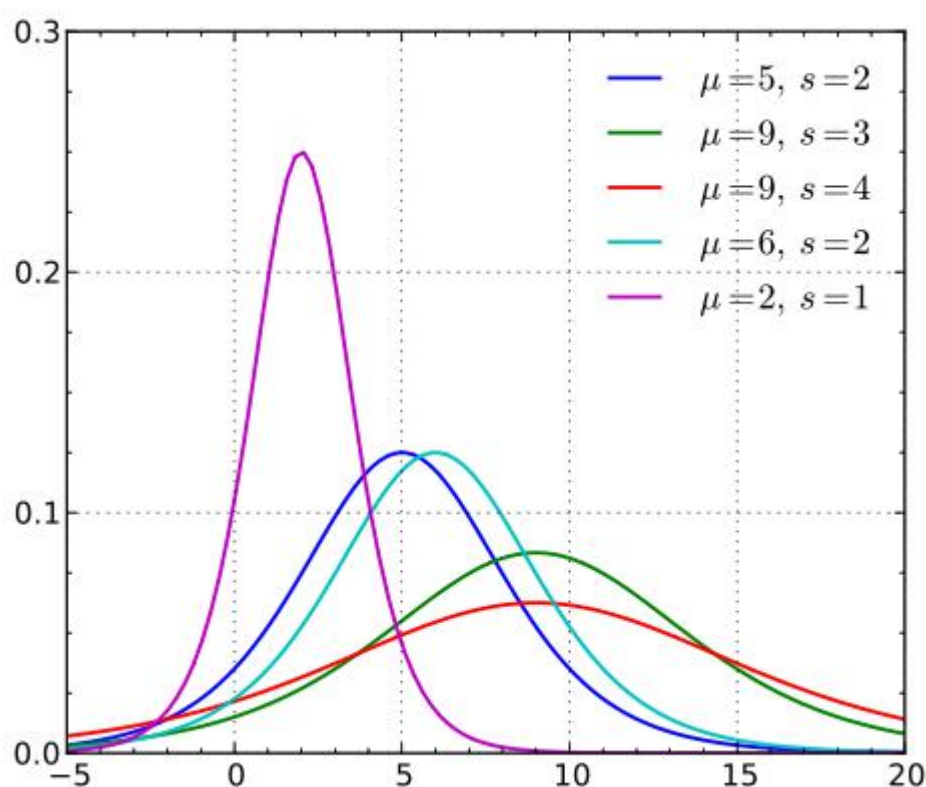
首先，需要对 logistic distribution （逻辑斯蒂分布）进行说明，这个分布的概率密度函数和概率分布函数如下：

$$p(x; \mu, s) = \frac{e^{-x-\mu/s}}{s(1 + e^{-x-\mu/s})^2}$$

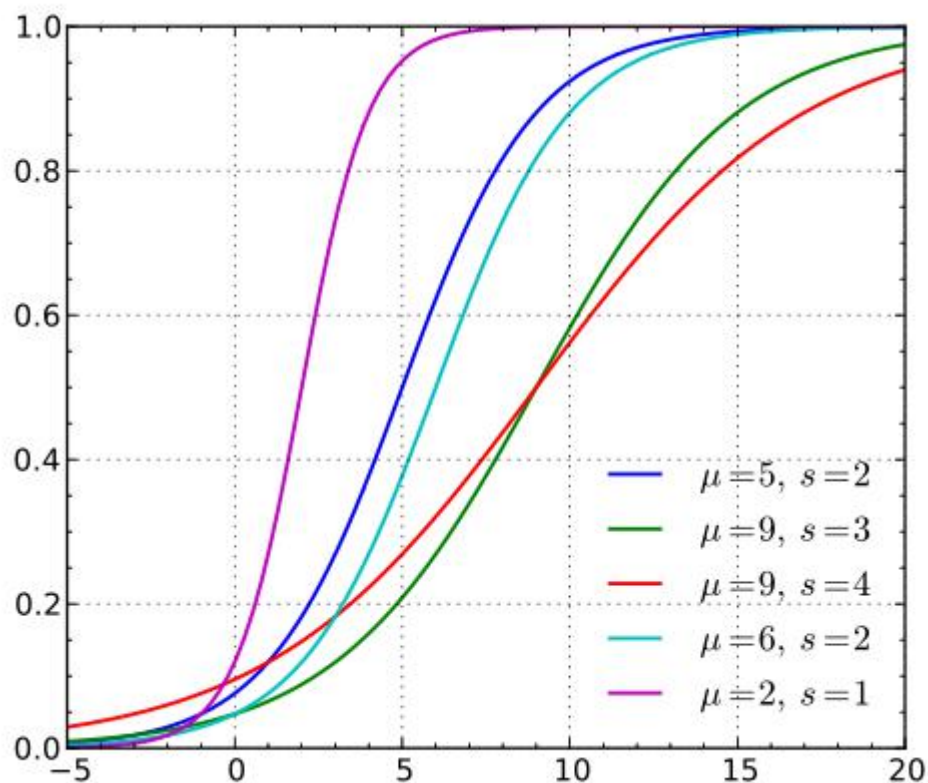
$$P(x; \mu, s) = \frac{1}{1 + e^{-x-\mu/s}}$$

这里 μ 是位置参数，而 s 是形状参数。

逻辑斯蒂分布在不同的 μ 和 s 的情况下，其概率密度函数 $p(x; \mu, s)$ 的图形：



逻辑斯蒂分布在不同的 μ 和 s 的情况下，其概率分布函数 $P(x; \mu, s)$ 的图形：



由图可以看出，逻辑斯蒂分布和高斯分布的密度函数长得差不多。特别注意逻辑斯蒂分布的概率密度函数自中心附近增长速度较快，而在两端的增长速度相对较慢。形状参数 s 的数值越小，则概率密度函数在中心附近增长越快。

当 $\mu = 0, s = 1$ 时，逻辑斯蒂分布的概率分布函数（累计密度函数）就是我们常说的 *sigmoid* 函数：

$$\sigma(p) = \frac{1}{1 + e^{-p}}$$

且其导数为：

$$\frac{d\sigma}{dp} = \sigma(1 - \sigma)$$

这是一个非常好的特性，并且这个特性在后面的推导中将会被用到。

3.从逻辑斯蒂分布 到 逻辑斯蒂回归模型

前面说过，分类算法都是求解 $p(C_k|x)$ ，而逻辑斯蒂回归模型，就是用当 $\mu = 0, s = 1$ 时的逻辑斯蒂分布的概率分布函数：*sigmoid* 函数，对 $p(C_k|x)$ 进行建模，所得到的模型，对于二分类的逻辑斯蒂回归模型有：

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)} = \frac{e^p}{1 + e^{-p}}$$

$$P(C_2|x) = \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)} = 1 - \frac{e^p}{1 + e^{-p}}$$

很容易求得，这里：

$$p = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$

这里 p 又可以被称作 $odds$ 比值比，又名机会比、优势比。（符号标记不慎，这里的 p 标记其实并不太合适。）

p 就是我们在分类算法中的决策面，由于 LR 回归是一个线性分类算法，所以，

此处采用线性模型：这里参数向量为 $\omega = (w_0, w_1, w_2, \dots, w_n)$ ， $\phi_j(x)$

是线性模型的基函数，基函数的数目为 M 个，如果定义 $\phi_0(x) = 1$ 的话。

$$h_\omega(x) = p = y(x, w) = \sum_{j=0}^M \omega_j \phi_j(x) = \omega^T \Phi(x)$$

$$w = (w_0, w_1, w_2, \dots, w_M)$$

$$\Phi = (\phi_0, \phi_1, \phi_2, \dots, \phi_M)$$

在这里的 p 其实就是 $h_\omega(x)$ （Hypothesis）。

那么，逻辑斯蒂回归模型 $p(C_k|x)$ 就可以重写为下面这个形式：

$$p(C_1|x) = \sigma(h_\omega(x)) = \sigma(\omega^T \Phi(x))$$

$$p(C_2|x) = 1 - \sigma(h_\omega(x)) = 1 - \sigma(\omega^T \Phi(x))$$

对于一个二分类的数据集 (x_n, t_n) ，这里 $t_n \in \{0, 1\}$ ，

$\Phi_n = \Phi(x_n)$ ， $t = (t_1, t_2, \dots)$ ， $y_n = p(C_1|x_n)$ 其极大似然估计为：

$$P(t|w) = P(t; w) = \prod_{n=1}^N y_n^{t_n} * (1 - y_n)^{(1-t_n)}$$

竖线改分号只是为了强调在这种情形下，它们含义是相同的，具体可见极大似然估计

对数似然估计为：（通常我们会取其平均 $\frac{1}{N}$ ）

$$\ln P(t|w) = \sum_{n=1}^N [t_n \ln y_n + (1 - t_n) \ln (1 - y_n)]$$

关于 w 的梯度为：

$$\frac{\partial \ln P(t|w)}{\partial w} = \sum_{n=1}^N \left[t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right]$$

得到梯度之后，那么就可以和线性回归模型一样，使用梯度下降法求解参数 w 。

$$w_{t+1} = w_t - \alpha \nabla_{\omega} \ln P(t|w)$$

梯度下降法实现相对简单，但是其收敛速度往往不尽人意，可以考虑使用随机梯度下降法来解决收敛速度的问题。但上面两种在最小值附近，都存在以一种曲折的慢速逼近方式来逼近最小点的问题。所以在 LR 回归的实际算法中，用到的是牛顿法，拟牛顿法（DFP、BFGS、L-BFGS）。

由于求解最优解的问题，其实是一个凸优化的问题，这些数值优化方法的区别仅仅在于选择什么方向走向最优解，而这个方向通常是优化函数在当前点的一阶导数（梯度）或者二阶导数（海森 Hessian 矩阵）决定的。比如梯度下降法用的就是一阶导数，而牛顿法和拟牛顿法用的就是二阶导数。

4.带惩罚项的 LR 回归

L2 惩罚的 LR 回归：

$$\min_w \left[\sum_{n=1}^N t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right] + \frac{\lambda}{2} w^T w$$

上式中， λ 是用于调节目标函数和惩罚项之间关系的， λ 越大，惩罚力度越大，所得到的 w 的最优解越趋近于 0，或者说参数向量越稀疏； λ 越小，惩罚力度越小，模型越复杂，也越能体现训练集的数据特征。

L1 惩罚的 LR 回归：

$$\min_w \left[\sum_{n=1}^N t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right] + \frac{\lambda}{2} \|w\|_1$$

$\|w\|_1$ 为 1 范数，即所有元素的绝对值之和。

L1 惩罚项可用于特征选择，会使模型的特征变得稀疏。

5.为什么逻辑回归比线性回归要好？

虽然逻辑回归能够用于分类，不过其本质还是线性回归。它仅在线性回归的基础上，在特征到结果的映射中加入了一层 sigmoid 函数（非线性）映射，即先把特

征线性求和，然后使用 sigmoid 函数来预测。然而，正是这个简单的逻辑函数，使得逻辑回归模型成为了机器学习领域一颗耀眼的明星。

这主要是由于线性回归在整个实数域内敏感度一致，而分类范围，只需要在 $[0, 1]$ 之内。而逻辑回归就是一种减小预测范围，将预测值限定为 $[0, 1]$ 间的一种回归模型。逻辑曲线在 $z=0$ 时，十分敏感，在 $z \gg 0$ 或 $z \ll 0$ 处，都不敏感，将预测值限定为 $[0, 1]$ 。

从梯度更新视角来看，为什么线性回归在整个实数域内敏感度一致不好。

线性回归和 LR 的梯度更新公式是一样的，如下：

$$\theta_j := 1/\alpha (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$$

唯一不同的是假设函数 $h_{\theta}(x^{(i)})$ ， $y \in [0, 1]$ ，对于 LR 而言， $h_{\theta}(x^{(i)}) \in [0, 1]$ ，那么梯度更新的幅度就不会太大。而线性回归 $h_{\theta}(x^{(i)})$ 在整个实数域上，即使已经分类正确的点，在梯度更新的过程中也会大大影响到其它数据的分类，就比如有一个正样本，其输出为 10000，此时梯度更新的时候，这个样本就会大大影响负样本的分类。而对于 LR 而言，这种非常肯定的正样本并不会影响负样本的分类情况！