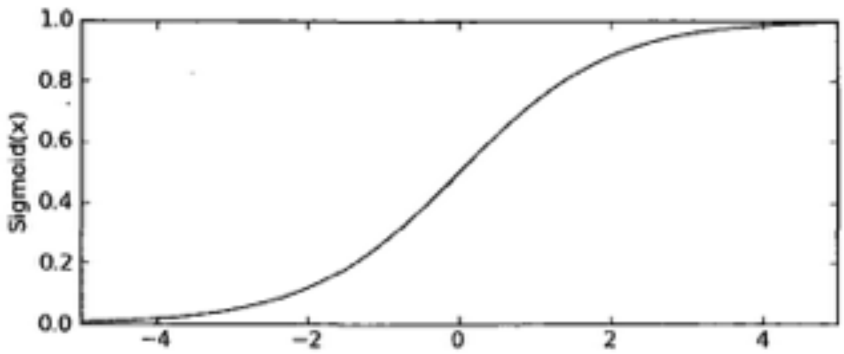
逻辑回归(LogisticRegression)

1. 简要介绍

一种广义线性模型，既可以做分类也可以做回归

累计分布函数曲线是一种S曲线

$$p(y=1|\mathbf{x};\theta)=\sigma(\theta^{\top}\mathbf{x})=\frac{1}{1+\exp(-\theta^{\top}\mathbf{x})}$$



当theta*X = 0时， p = 0.5;
当theta*X < 0时， p < 0.5，p(y = 1 | X;theta) < p(y = 0 | X;theta)， 预测为负样本
当theta*X > 0时， p > 0.5，p(y = 1 | X;theta) > p(y = 0 | X;theta)， 预测为正样本

定义一个线性函数 f(X) = theta*X，f(theta) = 0表示一个平面方程

当f(theta) < 0, 预测为负样本，待预测样本到平面方程的距离为abs(f(X)) / ||theta||
所以abs(f(X))越大， 距离就越大， 所以该样本被判定为负样本的可能性就越大
f(theta) > 0也是同样的分析。

所以我们要估计的是参数theta。

2. 参数估计

当正负样本标签为{0, 1}时, 可以使用极大似然估计法估计参数theta
单个样本概率为 :

$$p(y\,|\,x,\theta)=(h_{\theta}(x))^{y}(1-h_{\theta}(x))^{1-y}$$
$$\text{其中}y=1(\text{或}0)$$

对数似然函数

$$\begin{aligned}l(\theta)&=\log(L(\theta\,|\,x,y))\\&=\sum_{i=1}^{m}y^{(i)}\log(h(x^{(i)}))+(1-y^{(i)})\log(1-h(x^{(i)}))\end{aligned}$$

最大化L(theta)等价于最小化 -L(theta)

对目标函数求偏导：

$$\sum_{i=1}^{N} x_j * (h(x_i) - y_i)$$

所以使用梯度下降法迭代：

$$\Theta_j = \Theta_j - \alpha \sum_{i=1}^{N} x_j * (h(x_i) - y_i)$$

alpha表示学习率，alpha设的太小，收敛速度慢；设的太大， 可能会达不到最优。

3. 使用样例

```
# D. Update given model
# INPUT:
#     w: weights
#     n: a counter that counts the number of times we encounter a feature
#        this is used for adaptive learning rate
#     x: feature
#     p: prediction of our model
#     y: answer
# OUTPUT:
#     w: updated model
#     n: updated count
def update_w(w, n, x, p, y, pn, nn, p_num, n_num):
    for i in x:
        # alpha / (sqrt(n) + 1) is the adaptive learning rate heuristic
        # (p - y) * x[i] is the current gradient
        # note that in our case, if i in x then x[i] = 1
        n.setdefault(i, 0.)
        pn.setdefault(i, 0.)
        nn.setdefault(i, 0.)
        w[i] -= (p - y) * alpha / (sqrt(n[i]) + 1.)
        n[i] += 1.
        pn[i] += y
        nn[i] += 1-y
        p_num += y
        n_num += 1-y

    return w, n, pn, nn, p_num, n_num
```

```
map<int, double> instancefx;
for (int iter = 0; iter < opt.nr_iter; iter += 1)
{
        for (map<int, int>::iterator tit = trainDataLabel.begin(); tit != trainDataLabel.end(); ++tit)
    {
                int userid = tit->first;
                int yi = tit->second;

        double wxi = 0.0;
                map<int, double>& featuredict = trainDataFeature[userid];
        for (map<int, double>::iterator fit = featuredict.begin(); fit != featuredict.end(); ++fit)
        {
            int fid = fit->first;
            double fval = fit->second;
            wxi += g_featweightmap[fid] * fval;
        }
        double pi = 1 / (1 + exp(- wxi));
        double error = pi - yi;

        for (map<int, double>::iterator fit = featuredict.begin(); fit != featuredict.end(); ++fit)
        {
            int fid = fit->first;
            double fval = fit->second;
            g_featweightmap[fid] -= opt.nr_lr / pow(1 + iter, 1.0/3) * (error * fval + opt.nr_reg * g_featweightmap[fid]);
        }
    }
```

4.附录

附上损失函数的求偏导公式推导:

$$h = \frac{1}{1+e^{-\theta x}}$$

$$\underset{\theta}{\arg\min}\, L(\theta) = \sum_{i=1}^{N} -y_i \log h + (y_i - 1)\log(1-h)$$

$$\frac{\partial L}{\partial \theta_j} = \sum_{i=1}^{N} \frac{-y_i}{h} \cdot \frac{\partial h}{\partial \theta_j} + \frac{y_i - 1}{1-h} \cdot \left(-\frac{\partial h}{\partial \theta_j}\right)$$

$$= \sum_{i=1}^{N} -\frac{\partial h}{\partial \theta_j}\left(\frac{y_i}{h} + \frac{y_i - 1}{1-h}\right)$$

$$= \sum_{i=1}^{N} -\frac{\partial h}{\partial \theta_j} \cdot \frac{y_i(1-h) + h(y_i - 1)}{h(1-h)} \quad \text{而} \quad \frac{\partial h}{\partial \theta_j} = \frac{-x_j e^{-\theta x}}{-(1+e^{-\theta x})^2}$$

$$= x_j \cdot \frac{1}{1+e^{-\theta x}} \cdot \frac{e^{-\theta x}}{1+e^{-\theta x}}$$

$$= x_j \cdot h(1-h)$$

$$= \sum_{i=1}^{N} -x_j h(1-h) \cdot \frac{y_i - h}{h(1-h)}$$

$$= \sum_{i=1}^{N} x_j (h - y_i)$$

关于梯度下降法的一个小实验:
求凸函数f(x, y) = (x - 2) ^ 2 + (y - 1) ^ 2 + 1的最小值

```
#!/usr/bin/python
# -*- coding:utf8 -*-
import random
import numpy as np
import math
#f(x,y) = (x-2)^2+(y-1)^2 + 1
def solution(grad_func) :
    rate = 0.1
    x = random.uniform(-10,10)
    y = random.uniform(-10,10)
```

```
    point = np.array([x, y])
    for index in xrange(0, 1000) :
        grad = grad_func(point[0], point[1])
        point = point - rate * grad
        print grad
        if reduce(lambda a,b: math.sqrt(a*a+b*b), [grad[i] for i in xrange(grad.shape[0])]) < 0.000001 :
break
    print "times of iterate : %s" % index
    return point[0], point[1]
if __name__ == "__main__" :
    x, y = solution(lambda a,b: np.array([2*(a-2), 2*(b-1)]))
    print "minimum point of f(x,y) = (x-2)^2+(y-1)^2 + 1 : (%s,%s)" % (x, y)
```

执行结果:

```
[ 2.99291334  0.94751328]
[ 2.39433067  0.75801062]
[ 1.91546454  0.6064085 ]
[ 1.53237163  0.4851268 ]
[ 1.22589731  0.38810144]
… …
times of iterate : 68
minimum point of f(x,y) = (x-2)^2+(y-1)^2 + 1 : (2.0000003078,1.00000009745)
```

可以看到求得的点(2.0000003078,1.00000009745)与实际解(2, 1)是非常接近。