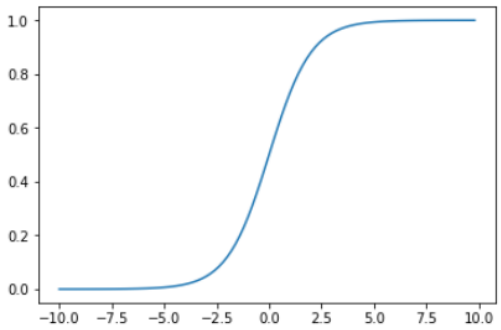
机器学习实验 4. Logistic 回归

Logistic 回归原理概述

Logistic 回归（对数几率回归/逻辑回归）是用于分类的回归方法。考虑二分类，其输出标记 $y \in \{0,1\}$，而线性回归模型产生的预测值 $z = w^T x + b$ 是实值，于是我们需要将 z 转换为 0/1 值。最理想的是单位阶跃函数：

$$y = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}。$$

但是该函数不连续，故而提出了 Sigmoid 函数 $\sigma(z) = \frac{1}{1+e^{-z}}$，其函数形式如下图：



虽然这里我们称为“Logistic 回归”，但是实际却是一种分类方法。由 z 的值可得到：

$$y = \frac{1}{1+e^{(-w^T x+b)}},$$

推导得 $\ln \frac{y}{1-y} = w^T x + b$。这里 $z = w^T x + b$ 虽然可以看作线性回归的结果，但在机器学习中更应该将其看作线性判别函数（决策面），其正好对应分类输出的判别结果（z 大于 0、小于 0 或等于 0），即将 $z = w^T x + b = g(x)$ 作为 sigmoid 函数的输入（图中 x 轴）。而在 sigmoid 函数中 y 并不是阶跃函数的形式，而是连续的平滑

函数，其意义可视为当前 x 具有正类的后验概率：$p(y=1|x)$，因此 logistic 回归也通常被看作一种软分类。此时，$\ln \frac{p(y=1|x)}{p(y=0|x)} = w^T x + b$，因此判别函数的判别结果（大于、小于、等于 0）分别意味着是正类的后验概率与负类的后验概率的比值。比值越大，说明 x 具有正类的后验概率越大，而对应的线性判别结果越大于 0；相反地，x 具有负类的后验概率越小，线性判别结果越小于 0；比值为 1，则正负类后验概率相等，此时线性判别结果恰好处于决策面上（$z = w^T x + b = 0$）。由此可以看出，logistic 实际上在用线性判别（回归）的结果去逼近真实标签的对数几率。

Logistic 回归的目标函数

由上述分析可知，Logistic 回归实际上是寻求线性决策（回归）结果与后验概率直接的对应关系，因此 Logistic 回归要求回归模型能使得每个样本属于其真实标签的概率越大越好。由此建立最大似然的目标函数：

$$L(W,b) = \prod_{i=1}^{m} p(y_i | x_i, W, b)$$

该似然函数反映对于 x，其隶属于正/负类概率，应分别与其真实标签尽可能对应。因此可以将上式进一步分解为正负两类情况，并让真实标签与对应后验概率相乘，从而优化使得样本具有真实标签的概率达到最大。

$$p(y_i | x_i, W, b) = y_i p(y_i = 1 | x_i, W, b) + (1 - y_i) p(y_i = 0 | x_i, W, b)$$

对似然函数取对数。由于 y_i 为 0 或 1 的二值输入，因此这里只对后验概率取对数，可得：

$$\ln L(W,b) = \sum_{i=1}^{m} \left[y_i \ln p(y_i = 1 | x_i, W, b) + (1 - y_i) \ln p(y_i = 0 | x_i, W, b) \right]$$

Logistic 回归的求解

将后验概率的 sigmoid 函数形式代入上式，并将 b 吸收到 W 中形成 β（$W;b$），此时 $x_i=(x_i;1)$：

$$\ln(L(\beta))=\sum_{i=1}^{m}\left[y_i\frac{e^{\beta^T x_i}}{1+e^{\beta^T x_i}}+(1-y_i)\frac{1}{1+e^{\beta^T x_i}}\right]$$

经过化简可得：

$$J(\beta)=-\max L(\beta)=\min\sum_{i=1}^{m}[-y_i\beta x_i+\ln(1+e^{\beta^T x_i})]$$

对上式使用牛顿迭代法求解最优解：

$$\beta^*=\arg\min_{\beta}J(\beta)$$

其第 t+1 轮迭代的更新公式为：

$$\beta^{t+1}=\beta^t-\left(\frac{\partial^2 J(\beta)}{\partial\beta\partial\beta^T}\right)^{-1}\frac{\partial J(\beta)}{\partial\beta}$$

关于 β 的一阶，二阶导数分别为：

$$\frac{\partial J(\beta)}{\partial\beta}=-\sum_{i=1}^{m}x_i(y_i-p(y_i=1\mid x_i,\beta))$$

$$\frac{\partial^2 J(\beta)}{\partial\beta\partial\beta^T}=\sum_{i=1}^{m}x_i x_i^T p_1(x_i,\beta)(1-p(y_i=1\mid x_i,\beta))$$

利用矩阵形式，上述一阶导数也可以改写为：$\frac{\partial J(\beta)}{\partial\beta}=X^T(p(Y=1\mid X,\beta)-Y)$，并利用梯度下降方法进行优化。

由此可见，Logistic 回归是这样的一个过程：面对一个回归或者分类问题，建立代价函数，然后通过优化方法迭代求解出最优的模型参数，然后测试验证我们这个求解的模型的好坏。我们想要得到的函数应该是能够接收所有的输入并且预测出的类别的。

Logistic 回归算法描述

（1）输入数据，并将其转换为结构化数据

（2）利用梯度上升函数求解参数 w

（3）基于训练好的回归系数 w 对结构化数据进行回归计算，判定其属于哪个类别

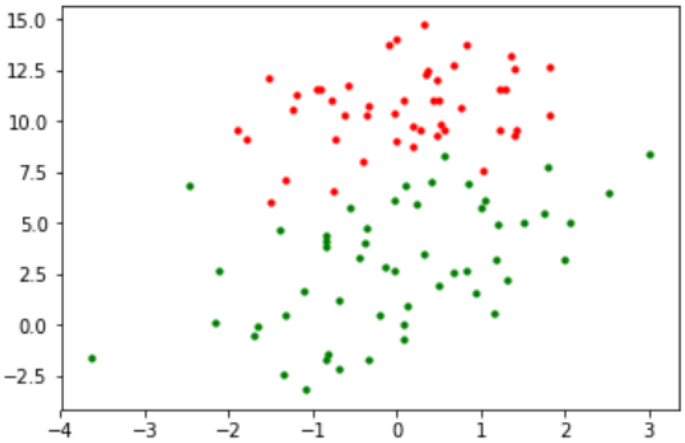
（4）将测试集放入

Logistic 回归的一般实现

1）生成数据集

```
from numpy import *
def loadDataSet():
    dataMat = []; labelMat = []
    fr = open('testSet.txt')
    for line in fr.readlines():
        lineArr = line.strip().split()
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
        labelMat.append(int(lineArr[2]))
    return dataMat,labelMat
```

使用 open 方法导入数据集，逐行读取。每行前两个值分别为 X1 和 X2，第三个值是数据对应的类标签



此时，由原本的数据生成这样的分布图。由图可见，红色点为一类，绿色点为一类。他们根据参数值分布在下面这张图上。我们要做的就是将这两种不同颜

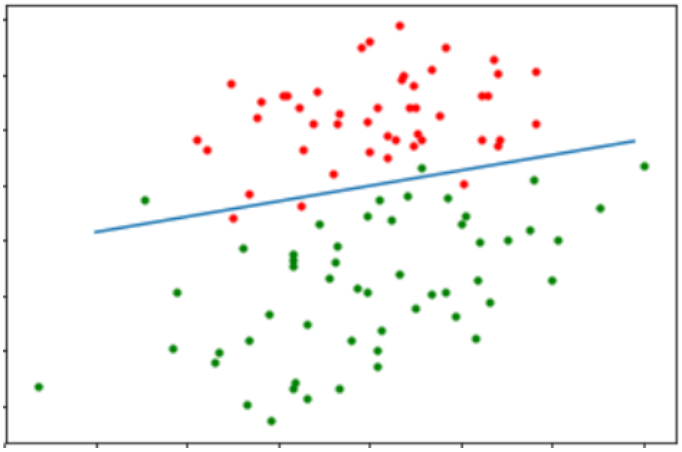
色得到点分开，找到一个可行的决策面，将下面这张图划分开，一部分属于红色的节点，另一部分属于绿色的节点。

2）利用梯度上升函数求参数 w

```
def gradAscent(dataMatIn, classLabels):
    dataMatrix = mat(dataMatIn)             #convert to NumPy matrix
    labelMat = mat(classLabels).transpose() #convert to NumPy matrix
    m,n = shape(dataMatrix)
```

梯度上升算法里有两个参数（dataMatIn，classLable）。其中 dataMatIn 是一个二维的 NumPy 数组，每列分别代表不同的特征，每行则代表每个训练样本。我们现在采用 100 个样本的简单数据集，它包含了两个特征，所以 dataMatIn 就是 100×3 的矩阵。classLable 是类别标签，是一个 1×100 的行向量。

3）画出决策边界



4）随机梯度上升算法

可以写成如下伪代码：

- 所有回归系数初始化为 1
  - 对数据集中的每个样本
  - 计算该样本的梯度
  - 重新计算更新回归系数
- 返回回归系数

```
def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.01
    weights = ones(n)    #initialize to all ones
    for i in range(m):
        h = sigmoid(sum(dataMatrix[i]*weights))
        error = classLabels[i] - h
        weights = weights + alpha * error * dataMatrix[i]
    return weights
```

（5）将测试集放入，比较所得到的决策面是否正确。