logistic 回归原理分析及分类应用

Y30180702　左春玲

一、logistic 回归和线性回归的关系

首先给出线性回归模型：

$$\mathrm{f}_w(\mathbf{x}) = w_0x_0 + w_1x_1 + \cdots + w_nx_n + \mathrm{b}$$

写成向量形式为：

$$\mathrm{f}_w(\mathrm{x}) = \boldsymbol{w}^T\boldsymbol{x}$$

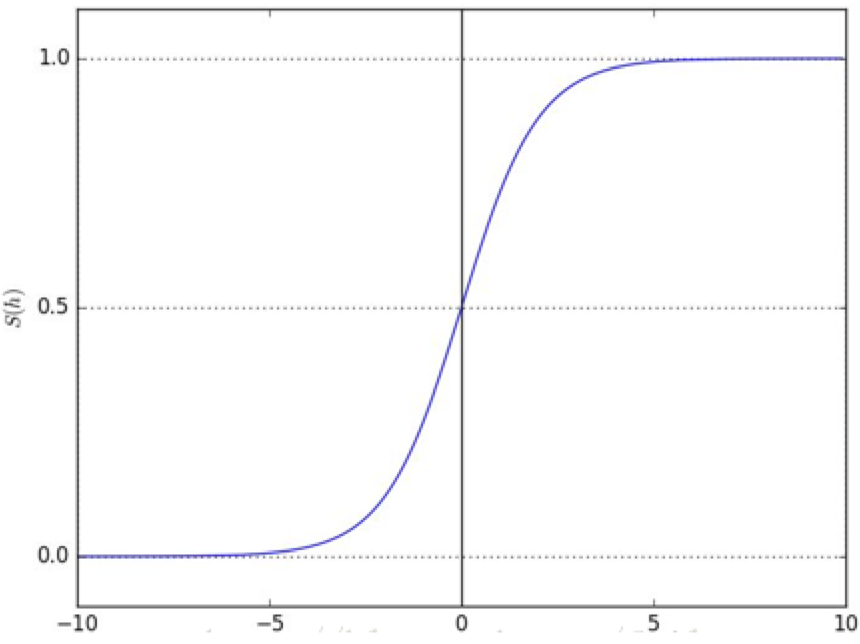
logistic 回归为二分类模型，利用 sigmoid 函数构造预测模型：

$$h_w(\mathbf{x}) = g(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

Sigmoid 函数为:

$$\mathrm{g(z)} = \frac{1}{1+e^{-z}}$$

图象如下所示：



我们在原来的线性回归模型外套上 sigmoid 函数便形成了 logistic 回

归模型的预测函数，可以用于二分类问题。

二、Logistic 回归模型

考虑具有 n 个独立变量的向量$\mathrm{x}=(x_1, x_2 \dots, x_n)$，构造预测模型：

$$h_w(\mathbf{x})=g(\boldsymbol{w}^T\boldsymbol{x})=\frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

设条件概率$\mathrm{P}(\mathrm{y}=1|\mathrm{x})=\mathrm{p}$为根据观测量相对于某事件 x 发生的概率为

$$\mathrm{P}(\mathrm{y}=1|\mathrm{x};\mathrm{w})=h_w(\mathbf{x})$$

那么在 x 条件下 y=0 的概率为

$$\mathrm{P}(\mathrm{y}=0|\mathrm{x};\mathrm{w})=1-\mathrm{P}(\mathrm{y}=1|\mathrm{x};\mathrm{w})=1-h_w(\mathbf{x})$$

综合起来即为

$$\mathrm{P}(\mathrm{y}|\mathrm{x};\mathrm{w})=\left(h_w(\mathbf{x})\right)^y(1-h_w(\mathbf{x}))^{1-y}$$

三、构造 cost 函数

取似然函数为

$$\mathrm{L}(\mathrm{w})=\prod_{i=1}^{m}P\left(y^{(i)}\middle|x^{(i)};w\right)$$
$$=\prod_{i=1}^{m}\left(h_w\left(x^{(i)}\right)\right)^{y^{(i)}}(1-h_w\left(x^{(i)}\right))^{1-y^{(i)}}$$

对数似然函数为：

$$\mathrm{l}(\mathrm{w})=\mathrm{logL}(\mathrm{w})$$
$$=\prod_{i=1}^{m}(y^{(i)}logh_w\left(x^{(i)}\right)+(1-y^{(i)})\log(1-h_w(x^{(i)})))$$

最大似然估计就是要求得使 l(w)取最大值时的 w，令

$$\mathrm{J}(\mathrm{w})=-\frac{1}{m}l(w)$$

乘了一个负的系数$-\frac{1}{m}$，所以 J(w)取最小时的 w 为要求的最佳参数，可以用梯度下降法求得。

四、梯度下降法求 J(w)的最小值

根据梯度下降法可得 w 的更新过程：

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w), (j = 0,1,\dots,n)$$

式中α为学习步长，求偏导如下：

$$\alpha \frac{\partial}{\partial w_j} J(w) =$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\frac{1}{h_w(x^{(i)})}\frac{\partial}{\partial w_j}h_w\left(x^{(i)}\right) - (1-y^{(i)})\frac{1}{1-h_w(x^{(i)})}\frac{\partial}{\partial w_j}h_w\left(x^{(i)}\right))$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\frac{1}{g(w^T x^{(i)})} - \left(1-y^{(i)}\right)\frac{1}{1-g(w^T x^{(i)})})\frac{\partial}{\partial w_j}g(w^T x^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\frac{1}{g(w^T x^{(i)})} - \left(1-y^{(i)}\right)\frac{1}{1-g(w^T x^{(i)})})g(w^T x^{(i)})(1-g(w^T x^{(i)}))\frac{\partial}{\partial w_j}w^T x^{(i)}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\left(1-g\left(w^T x^{(i)}\right)\right) - \left(1-y^{(i)}\right)g\left(w^T x^{(i)}\right))\, {x_j}^{(i)}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - g\left(w^T x^{(i)}\right)){x_j}^{(i)}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h_w\left(\boldsymbol{X}^{(i)}\right)){x_j}^{(i)}$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_w\left(\boldsymbol{X}^{(i)}\right) - y^{(i)}){x_j}^{(i)}$$

上式求解过程中用到如下的公式：

$$\mathrm{f(x)} = \frac{1}{1+e^{\mathrm{g}(x)}}$$

$$\frac{\partial}{\partial x}\mathrm{f(x)} = \frac{1}{(1+e^{g(x)})^2}e^{g(x)}\frac{\partial}{\partial x}g(x)$$

$$= \frac{1}{1+e^{g(x)}}\frac{e^{g(x)}}{1+e^{g(x)}}\frac{\partial}{\partial x}g(x)$$

$$= \mathrm{f(x)}(1-\mathrm{f(x)})\frac{\partial}{\partial x}g(x)$$

因此，w 的更新过程可以写成：

$$w_j \coloneqq w_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_w\left(\boldsymbol{X}^{(i)}\right) - y^{(i)})x_j{}^{(i)}, (j=0,1,\dots,n)$$

因为式中α本来为一常量，所以

$$w_j \coloneqq w_j - \alpha\sum_{i=1}^{m}\left(h_w\left(\boldsymbol{X}^{(i)}\right) - y^{(i)}\right)x_j{}^{(i)}, (j=0,1,\dots,n)$$

五、梯度下降过程向量化

首先对于引入的数据集 **x** 来说，均是以矩阵的形式引入的，**x** 的每一行为一条训练样本，而每一列为不同的特征值，如下：

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{x}^{(m)} \\ \vdots \\ \boldsymbol{x}^{(m)} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

其中 m 是数据样本的个数，n 是数据的维度，也就是数据特征的数量。

约定待求的参数 **w** 的矩阵形式为：

$$\mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$$

先求**xw** 并记为 **A**：

$$\mathbf{A}=\mathbf{x}\mathbf{w}=\begin{bmatrix}x_0^{(1)} & \cdots & x_n^{(1)}\\ \vdots & \ddots & \vdots\\ x_0^{(m)} & \cdots & x_n^{(m)}\end{bmatrix}\begin{bmatrix}w_0\\ \vdots\\ w_n\end{bmatrix}=\begin{bmatrix}w_0x_0^{(1)}+\ldots+w_nx_n^{(1)}\\ \vdots\\ w_0x_0^{(m)}+\ldots+w_nx_n^{(m)}\end{bmatrix}$$

求$\mathbf{h}_w(\mathbf{x})-\mathbf{y}$并记为 **E**：

$$\mathbf{E}=\mathbf{h}_w(\mathbf{x})-\mathbf{y}=\begin{bmatrix}g\left(A^{(1)}\right)-y^{(1)}\\ \vdots\\ g\left(A^{(m)}\right)-y^{(m)}\end{bmatrix}=\begin{bmatrix}e^{(1)}\\ \vdots\\ e^{(m)}\end{bmatrix}=\boldsymbol{g}(\boldsymbol{A})-\boldsymbol{y}$$

g(**A**)的参数 **A** 为一列向量，所以实现 **g(~)**函数时要支持列向量作为参数，并返回列向量。由上式可知$\mathbf{E}=\mathbf{h}_w(\mathbf{x})-\mathbf{y}$可以由$\boldsymbol{g}(\boldsymbol{A})-\boldsymbol{y}$一次计算求得。

再看 w 的更新过程，当 j=0 时：

$$w_0 \coloneqq w_0-\alpha\sum_{i=1}^{m}\left(h_w\left(\boldsymbol{X}^{(i)}\right)-y^{(i)}\right){x_0}^{(i)}$$
$$=w_0-\alpha\sum_{i=1}^{m}\left(e^{i}\right){x_0}^{(i)}$$
$$=w_0-\alpha\left({x_0}^{(1)},{x_0}^{(2)},\ldots,{x_0}^{(m)}\right)\boldsymbol{E}$$

同时可以求出w_j，

$$w_j:=w_j-\alpha\left({x_j}^{(1)},{x_j}^{(2)},\ldots,{x_j}^{(m)}\right)\boldsymbol{E}$$

综合起来就是：

$$\begin{bmatrix}\mathrm{w}_0\\ \vdots\\ \mathrm{w}_\mathrm{n}\end{bmatrix}\coloneqq\begin{bmatrix}w_0\\ \vdots\\ w_n\end{bmatrix}-\alpha\begin{bmatrix}{x_0}^{(1)},{x_0}^{(2)},\ldots,{x_0}^{(m)}\\ \vdots\\ {x_j}^{(1)},{x_j}^{(2)},\ldots,{x_j}^{(m)}\end{bmatrix}\boldsymbol{E}$$

$$\mathbf{w}:=\mathbf{w}-\alpha\boldsymbol{x}^{T}\boldsymbol{E}$$

下面便可以在实例中应用此迭代公式进行实际的分析了，下面便以简单的 iris 数据集的二分类问题为例来分析 logistic 回归算法的使用。

Python 程序如下：

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import datetime
from sklearn import preprocessing
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris

def loadDataSet():      #读取数据集并处理函数
    dataMatrix = []
    datalabel = []
    plt.style.use('ggplot')   #使用自带的样式进行美化
    iris = load_iris()          #获取数据
    data = iris.data
    target = iris.target
    X = data[0:100]              #选择前两类各 50 组数据
    Y = target[0:100]
    datalabel = np.mat(Y)       #使数据变成标准矩阵形式
    datalabel=np.transpose(datalabel)   #进行转置
    dataMatrix = np.mat(X)
```

```
        minmax_x_train = MinMaxScaler()
        x_train_std = minmax_x_train.fit_transform(dataMatrix)   #使数据标准化
        dataMatrix = np.mat(x_train_std)
        return dataMatrix,datalabel

    def sigmoid(X):
        return 1.0/(1+np.exp(-X))     #sigmoid 函数形式

    def graAscent(dataMatrix,matLabel,num):      #梯度下降
        m,n=np.shape(dataMatrix)
        w=np.ones((n,1))           #将 w 初始化为 1
        alpha=0.01                 #设置学习速率为 alpha
        for i in range(num):       #num 为迭代次数
            error=sigmoid(dataMatrix*w)-matLabel
            w=w-alpha*dataMatrix.transpose()*error
        return w

    def predict(w,X):              #预测函数
        m = X.shape[0]             #取列数
        Y_prediction = np.zeros((m,1))
        #w = w.ones(X.shape[0],1)
```

```
    A = sigmoid(np.dot(X,w))
    for i in range(A.shape[0]):
        if A[i,0]>0.5:
            Y_prediction[i ,0]=1
        else:
            Y_prediction[i ,0]=0
    assert(Y_prediction.shape == (m,1))
    return Y_prediction

def loss(X,Y,num,print_cost=False):     #损失函数
   #costs=loss(weight,dataMatrix,matLabel, num)
    m, n = np.shape(dataMatrix)
    w = np.ones((n, 1))
    alpha = 0.01
    costs = []
    print_cost=0
    for i in range(num):
        #  记录损失
        error = sigmoid(dataMatrix * w) - matLabel       #损失误差
        w = w - alpha * dataMatrix.transpose() * error   #更新 w
        A = sigmoid(np.dot(X, w))
        w = np.array(w)
```

```
        A = np.array(A)
        Y= np.array(Y)
        cost = (- 1 / m) * np.sum(Y * np.log(A) + (1 - Y) * (np.log(1 - A)))
        if i % 10== 0:
            costs.append(cost)
            print("迭代的次数: %i ， 误差值： %f" % (i, cost))
    return costs

if __name__ == '__main__':
    dataMatrix,matLabel=loadDataSet()
    print(dataMatrix.shape)     #打印训练数据维度
    print(matLabel.shape)        #打印训练目标维度
    num = 2000                          #设置迭代次数
    #weight=graAscent(dataMatrix,matLabel)
    weight= graAscent(dataMatrix,matLabel,num)    #计算权值
    print(weight.shape)            #打印权值矩阵维度
    y=predict(weight,dataMatrix)    #训练数据的预测结果
    print(y.T)
    print("训练集的准确度为：", format(100 - np.mean(np.abs(y - matLabel) * 100)), "%")
    costs = loss(dataMatrix, matLabel, num)
```

```
plt.plot(costs)

plt.ylabel('cost')

plt.xlabel('iterations (per hundreds)')

plt.show()
```

结果为：

```
(100, 4)
(100, 1)
(4, 1)
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
  0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1.
  1. 1. 1. 1.]]
训练集的准确度为： 100.0 %
```