AI学习笔记--skLearn--Logistic

- 概述

logistic回归又称logistic回归分析，主要在流行病学中应用较多，比较常用的情形是探索某疾病的危险因素，根据危险因素预测某疾病发生的概率，等等。例如，想探讨胃癌发生的危险因素，可以选择两组人群，一组是胃癌组，一组是非胃癌组，两组人群肯定有不同的体征和生活方式等。这里的因变量就是--是否胃癌，即“是”或“否”，为两分类变量，自变量就可以包括很多了，例如年龄、性别、饮食习惯、幽门螺杆菌感染等。自变量既可以是连续的，也可以是分类的。通过logistic回归分析，就可以大致了解到底哪些因素是胃癌的危险因素。

- 算法原理

logistic回归(Logistic regression) 与多重线性回归实际上有很多相同之处，最大的区别就在于他们的因变量不同，其他的基本都差不多，正是因为如此，这两种回归可以归于同一个家族，即广义线性模型（generalized linear model）。这一家族中的模型形式基本上都差不多，不同的就是因变量不同，如果是连续的，就是多重线性回归，如果是二项分布，就是logistic回归，如果是poisson分布，就是poisson回归，如果是负二项分布，就是负二项回归，等等。只要注意区分它们的因变量就可以了。[1]

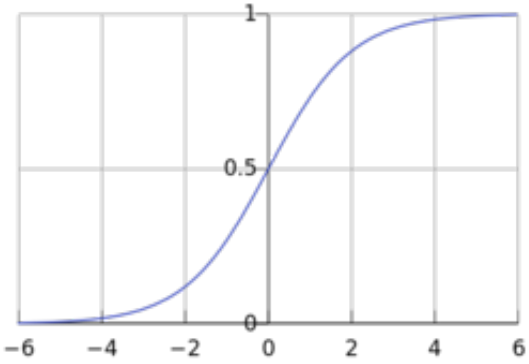
logistic回归的因变量可以是二分非线性差分方程类的，也可以是多分类的，但是二分类的更为常用，也更加容易解释。所以实际中最为常用的就是二分类的logistic回归。

Logistic函数：

对于任意的x值，对应的y值都在区间(0,1)内。函数公式为：

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

这个函数的曲线大致如下图所示：



为了实现Logistic回归，我们可以在已有数据的每个特征值上都乘以一个回归系数，然

后把所有的值相加，将这个总和代入sigmond函数，进而得到一个范围在0~1之间的数值。任何大于0.5的数据被映射为1，小于0.5的数据被映射为0。

如果说线性回归是对于特征的线性组合来拟合真实标记的话（y=wx+by=wx+b），那么逻辑回归是对于特征的线性组合来拟合真实标记为正例的概率的对数几率（lny1−y=wx+blny1−y=wx+b）。对于输入xx分类结果为类别1和类别0的概率分别为：

$$P(y=1|x;\theta)=h_\theta(x)$$

$$P(y=0|x;\theta)=1-h_\theta(x).$$

例如，如果对于给定的x，通过已经确定的参数计算得出hθ(x)=0.7，则表示有70%的几率y为正例，相应的y为负例的几率为0.3。

- **决策面**

在逻辑回归中，我们预测：

- 当h_θ大于等于0.5时，预测y=1；
- 当h_θ小于0.5时，预测y=0。

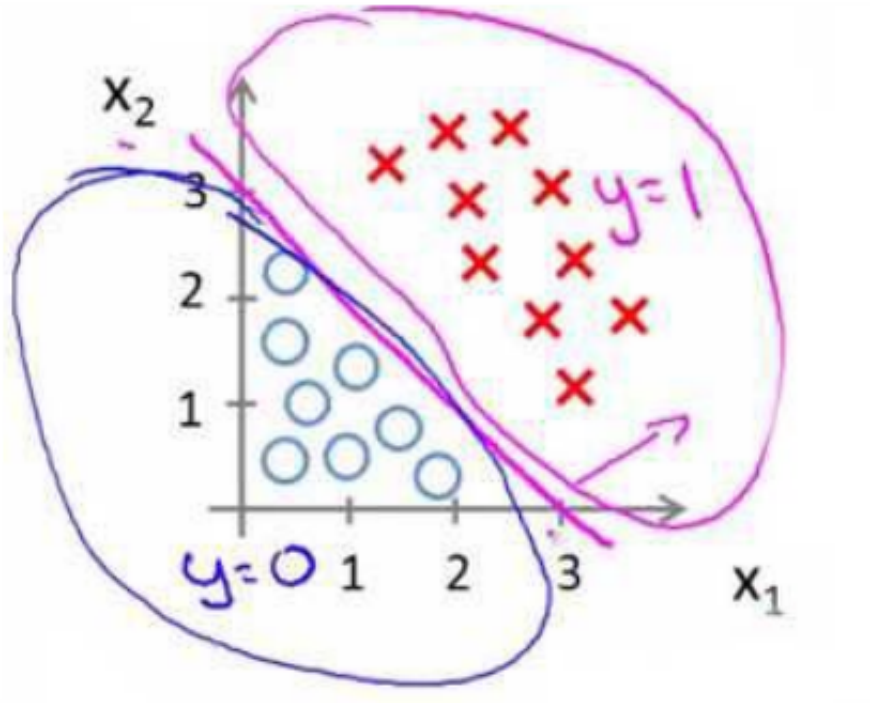
根据上面绘制出的S形函数图像，我们知道：

- $z=0$时$g(z)=0.5$,
- $z>0$时$g(z)>0.5$,
- $z<0$时$g(z)<0.5$

因为$z=\theta^T x$ ，所以满足：

- $\theta^T x$大于等于0时，预测$y=1$；
- $\theta^T x$小于0时，预测$y=0$。

假设我们有一个模型：$h_\theta(x)=g(\theta_0+\theta_1 x_1+\theta_2 x_2)$,并且参数θ 满足的向量模型是[-3 1 1]则当$-3+x_1+x_2$ 大于等于0是，即x_1+x_2 大于3时，我们预测y=1 ,由此可以得到，$x_1+x_2=3$是这个模型的分界线。如下图所示：

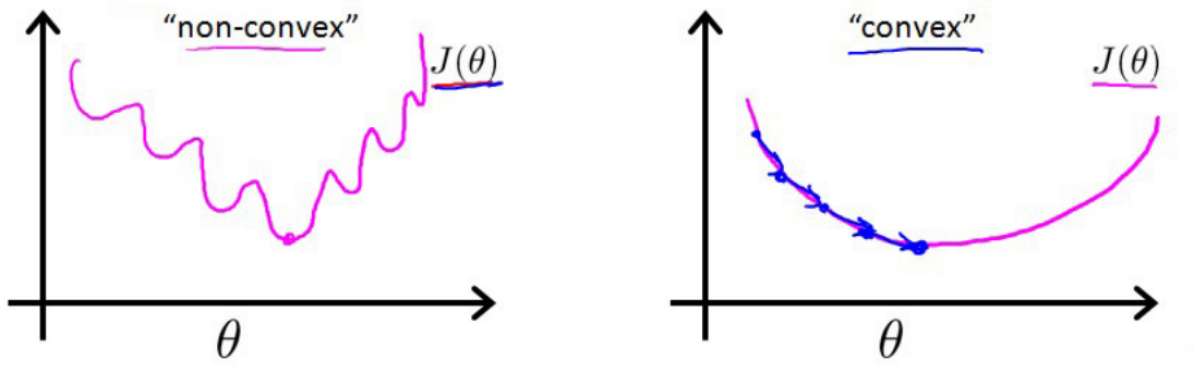


- 损失函数

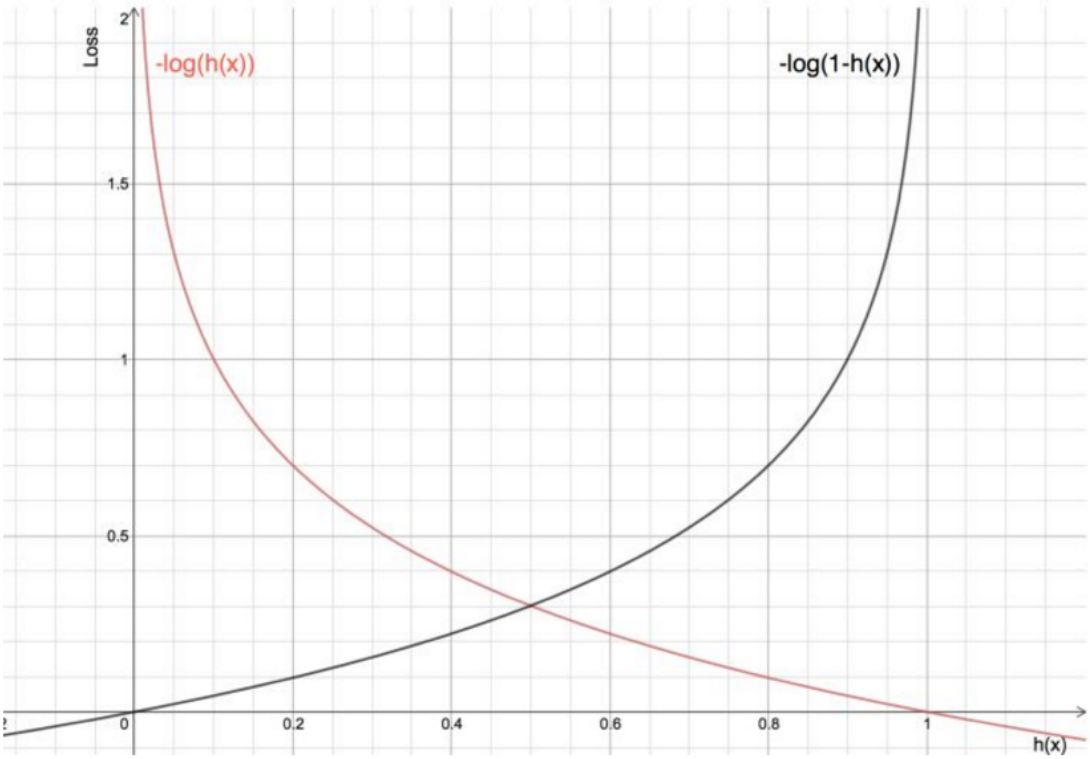
如何求分类任务的最优解呢？第一个直接的想法是仍然沿用上述的均方误差来表示真实样本与预测值之间的差距,公式：

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(\frac{1}{1+e^{-\theta^T x^{(i)}-b}} - y^{(i)})^2$$

但这个函数不是凸函数，很难进行优化。



对于真实标记是1的样本，我们希望预测值越接近于1，损失越小；对于真实标记是0的样本，我们希望预测值越接近于0时损失越小，-log函数正好满足以上情况：



1

Andrew Ng在课程中直接给出了交叉熵损失Cost函数及数据集全部损失J(θ)函数，但是并没有给出具体的解释，只是说明了这个函数来衡量hθ(x)函数预测的好坏是合理的。

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases}$$

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta(x^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right] \end{aligned}$$

所以最后目标变成取J(θ)最小值时的θ为最佳参数。与线性回归类似，利用梯度下降法更新θ

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial_{\theta_j}}J(\theta) = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})x_j^{(i)}$$

- 算法Demo

plotBestFit.py

```
# -*- coding:utf-8 -*-

import matplotlib.pyplot as plt
```

```
from numpy.ma import arange


# 描绘最佳拟合直线
def plotBestFit(w, b, dataMat, labelMat):
    m = dataMat.shape[0]
    xcord1 = []
    ycord1 = []
    xcord2 = []
    ycord2 = []
    for i in range(m):
        if int(labelMat[i]) == 1:
            xcord1.append(dataMat[i, 0])
            ycord1.append(dataMat[i, 1])
        else:
            xcord2.append(dataMat[i, 0])
            ycord2.append(dataMat[i, 1])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(xcord1, ycord1, s=30, c='red', marker='x')
    ax.scatter(xcord2, ycord2, s=30, c='green', marker='o')
    x = arange(-3.0, 3.0, 0.1)
    y = (-w[0] * x - b) / w[1]
    ax.plot(x, y)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
```

主类:

```
# -*- coding:utf-8 -*-

import numpy as np
from plotBestFit import plotBestFit


# sigmoid函数
def sigmoid(z):
    s = 1.0 / (1 + np.exp(-z))
    return s


# 计算梯度和损失值
def calculate_grads_cost(w, b, X, Y):
    m = X.shape[0]

    A = sigmoid(np.dot(X, w) + b)
    cost = -np.sum(Y * np.log(A) + (1 - Y) * np.log(1 - A)) / m

    dw = np.dot(X.T, A - Y)
```

```
    db = np.sum(A - Y)

    grads = {
        "dw": dw,
        "db": db
    }

    return grads, cost


# 使用梯度下降优化cost
def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost=False):
    costs = []

    for i in range(num_iterations):
        grads, cost = calculate_grads_cost(w, b, X, Y)

        dw = grads["dw"]
        db = grads["db"]

        w = w - learning_rate * dw
        b = b - learning_rate * db

        if i % 100 == 0:
            costs.append(cost)

        if print_cost and i % 100 == 0:
            print("第 %d 轮迭代后的损失值为:  %f" % (i, cost))

    params = {
        "w": w,
        "b": b
    }

    grads = {
        "dw": dw,
        "db": db
    }

    return params, grads, costs


if __name__ == "__main__":
    data = np.loadtxt("testSet.txt")
    dataMat = data[:, 0:2]
    labelMat = data[:, 2]
    m, n = dataMat.shape
    labelMat = labelMat.reshape(m, 1)
    w = np.zeros((n, 1))
    b = 0
    params, grads, costs = optimize(w, b, dataMat, labelMat, 200, 0.1)
```

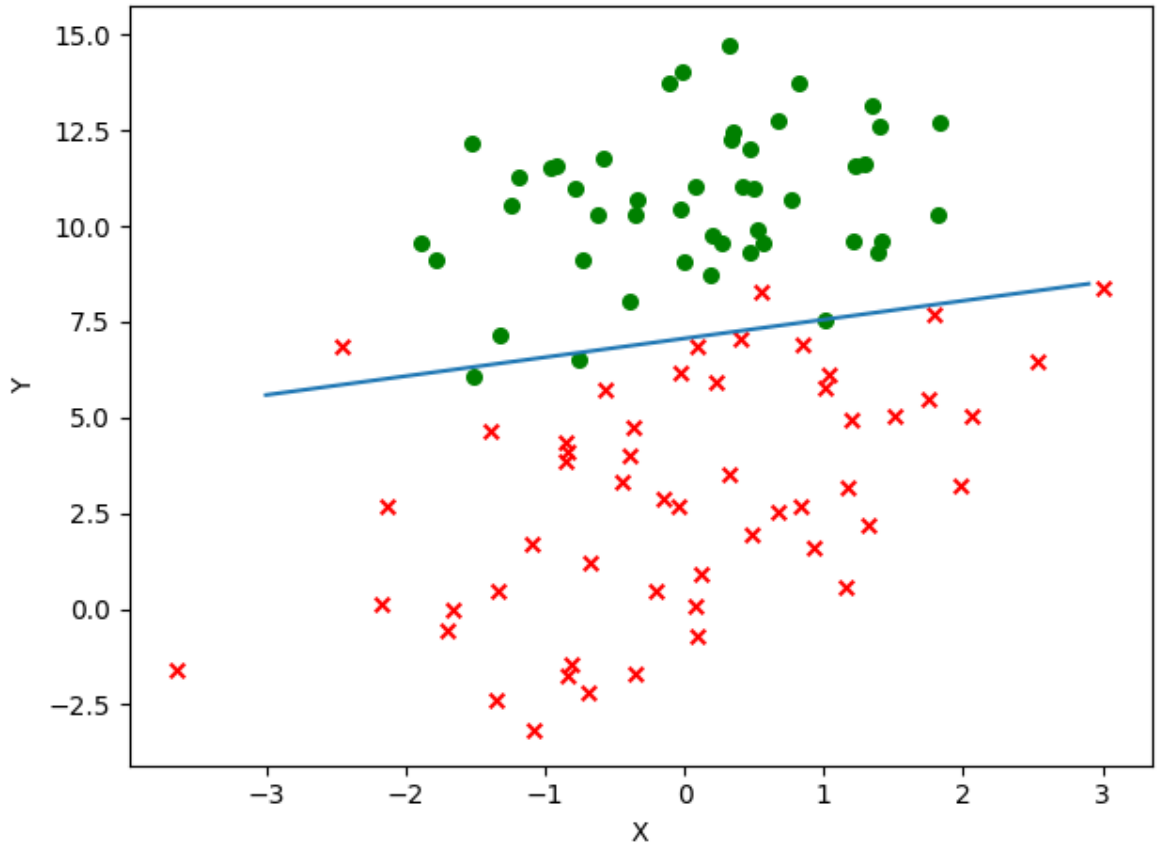
```
    w = params["w"]
    b = params["b"]
    # print(w)
    # print(b)

    # 可视化结果
    plotBestFit(w, b, dataMat, labelMat)
```

运行数据：

testSet.txt

运行结果：



Sk-Learn Demo:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
=========================================================
Logistic Regression 3-class Classifier
=========================================================
```

```
Show below is a logistic-regression classifiers decision boundaries on the
first two dimensions (sepal length and width) of the `iris
<https://en.wikipedia.org/wiki/Iris_flower_data_set>`_ dataset. The
datapoints
are colored according to their labels.

"""
print(__doc__)

# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

logreg = LogisticRegression(C=1e5, solver='lbfgs',
multi_class='multinomial')

# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X, Y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,
h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
```

```
plt.yticks(())

plt.show()
```