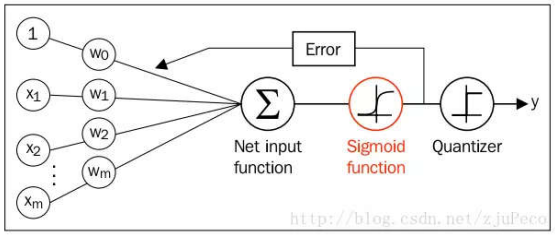
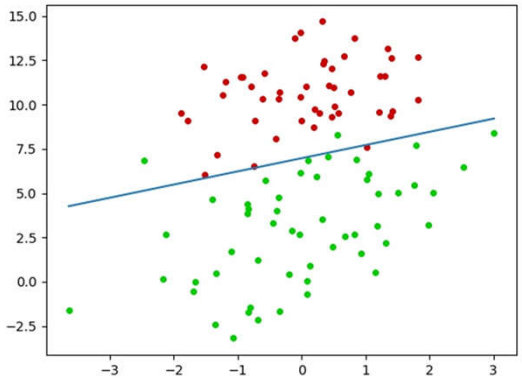
Python编程与人工智能实践





算法篇：逻辑回归（logistic regression）

于泓
鲁东大学
信息与电气工程学院

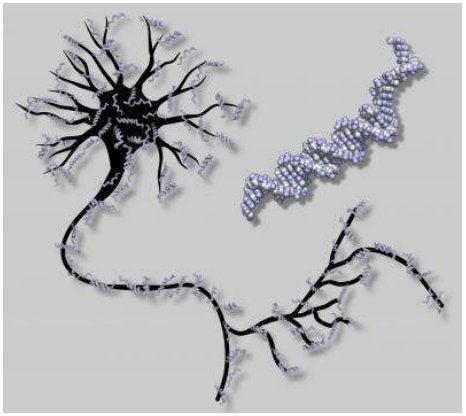
2021.4.1

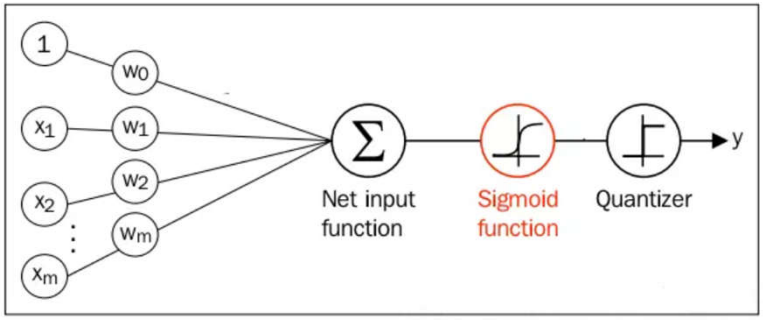
逻辑回归（logistic regression）

- 回归：假设有一些数据点。利用一条直线对这些点进行拟合就叫做回归
- 逻辑回归：回归的目标是一个二值的结果（0,1），是最常用的一种分类模型。

已知一组数据**X** 以及其相应的二值标签 Y
利用这些数据构造一个函数，预测新的输入X_i的标签

X_1 ----→Y_1（0）
X_2 ----→Y_2（1）
X_3----→Y_3（1）
X_4 ----→Y_4（0）
X_5 ----→Y_5（1）
X_6 ----→Y_6（0）
X_7 ----→Y_7（1）
……
X_N ----→Y_N（1）





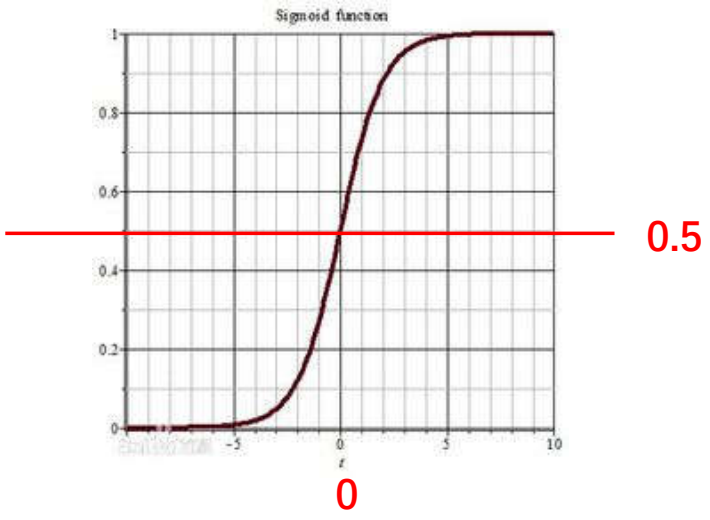
假设X是一个2维的特征

$$h(X;\mathbf{w})=\text{sigmoid}(x_2*\text{w}_2+x_1*\text{w}_1+\text{w}_0)=\text{sigmoid}(X\mathbf{w})$$

其中

$$\sigma(\text{z})=\text{sigmoid}(\text{z})=\frac{1}{1+\text{e}^{-\text{z}}}$$

输出范围[0,1],可以把输出理解为Y=1的概率
概率>0.5 输出标签1，否则输出0



已知输入特征X_i，　**预测**输出标签$Y_i=1$ 的概率为:

$$P(Y_i=1|X_i;\mathbf{w})=h(X_i;\mathbf{w})=\sigma(X_i\mathbf{w})$$

已知输入特征X_i，　**预测**输出标签$Y_i=0$ 的概率为:

$$P(Y_i=0|X_i;\mathbf{w})=1-h(X_i;\mathbf{w})$$

已知输入特征X_i以及对应的标签Y_i
预测准确的概率为:

$$P(Y_i,X_i;\mathbf{w})=\left[h(X_i;\mathbf{w})\right]^{Y_i}\left[1-h(X_i;\mathbf{w})\right]^{1-Y_i}$$

预测的越准 上述函数就越大

根据已知数据，寻找最优参数 **w**，就是令

$$L(\mathbf{w})=\prod_{i=1}^{N}P(Y_i,X_i;\mathbf{w})$$

$$=\prod_{i=1}^{N}\left[h(X_i;\mathbf{w})\right]^{Y_i}\left[1-h(X_i;\mathbf{w})\right]^{1-Y_i}$$ **最大**

取log写成似然函数的形式

$$L(\mathbf{w})=\sum_{i=1}^{N}Y_i\log\left(h(X_i;\mathbf{w})\right)+\left(1-Y_i\right)\log\left(1-h(X_i;\mathbf{w})\right)$$

最大

再加一个**“-”号上式就变成了了熵的形式（交叉熵）**

$$L(\mathbf{w})=-\sum_{i=1}^{N}Y_i\log\left(h(X_i;\mathbf{w})\right)+\left(1-Y_i\right)\log\left(1-h(X_i;\mathbf{w})\right)$$

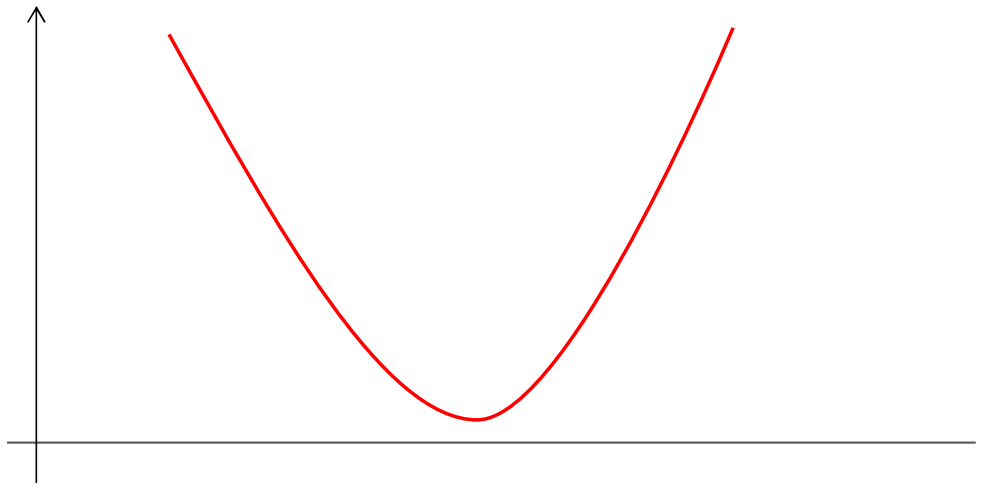
熵越小系统越确定,分类也就越准确

逻辑回归的最终目标变为：
已知训练**数据X**以及相应**标签Y**
求解**参数w**使 **交叉熵*L*(w)** **最小**

$$L(\mathbf{w})=-\sum_{i=1}^{N} Y_i \ln\left(h(X_i;\mathbf{w})\right)+\left(1-Y_i\right)\ln\left(1-h(X_i;\mathbf{w})\right)$$

上述方程没有闭式解需要通过**梯度下降**的方法计算最优解。

$$\mathbf{w}_{\text{new}}=\mathbf{w}_{\text{old}}-\alpha\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$



令 $z_i=X_i\mathbf{w}$

$$L(\mathbf{w})=-\sum_{i=1}^{N}Y_i\ln\left(h(X_i;\mathbf{w})\right)+\left(1-Y_i\right)\ln\left(1-h(X_i;\mathbf{w})\right)$$

$$=-\sum_{i=1}^{N}Y_i\ln\frac{h(z_i)}{1-h(z_i)}+\ln(1-h(z_i))$$

$$=-\sum_{i=1}^{N}z_iY_i-z_i-\ln(1+e^{-z_i})$$

$$\frac{\partial L(\mathbf{w})}{\partial\mathbf{w}}=\frac{\partial L(\mathbf{w})}{\partial z}\frac{\partial z}{\partial\mathbf{w}}=-\sum_{i=1}^{N}\left(Y_i-1-\frac{-e^{-z_i}}{1+e^{-z_i}}\right)x_i^{\mathrm{T}}$$

$$=-\sum_{i=1}^{N}\left(Y_i-\frac{1}{1+e^{-z_i}}\right)x_i=-\sum_{i=1}^{N}\left(Y_i-h(x_i;\mathbf{w})\right)x_i^{\mathrm{T}}$$

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

$$\ln(1-h(z))=\ln\left(1-\frac{1}{1+e^{-z}}\right)=\ln\frac{e^{-z}}{1+e^{-z}}=-z-\ln(1+e^{-z})$$

代入

标签　　实际输出

$$\mathbf{w}_{\text{new}}=\mathbf{w}_{\text{old}}-\alpha\frac{\partial L(\mathbf{w})}{\partial\mathbf{w}}=\mathbf{w}_{\text{old}}+\alpha\sum_{i=1}^{N}\left(Y_i-h(x_i;\mathbf{w})\right)x_i^{\mathrm{T}}$$

写成矩阵形式

$$\mathbf{w}_{\text{new}}=\mathbf{w}_{\text{old}}+\alpha\mathbf{X}^{\mathrm{T}}\left(\mathbf{Y}-h(\mathbf{X};\mathbf{w}_{\text{old}})\right)$$

$$\mathbf{X}\in[N\times D]$$

$$\mathbf{Y}\in[N\times 1]$$

$$\mathbf{w}\in[D\times 1]$$

定义sigmoid 函数

```python
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(z):
    return 1.0/(1+np.exp(-z))
```

$$\mathbf{w}_{new}=\mathbf{w}_{old}+\alpha\mathbf{X}^{T}\left(\mathbf{Y}\text{-}h(\mathbf{X};\mathbf{w}_{old})\right)$$

$$\mathbf{X}\in[N\times D]$$

$$\mathbf{Y}\in[N\times 1]$$

$$\mathbf{w}\in[D\times 1]$$

```python
def weight_update(datas,labs,w,alpha=0.01):
    z = np.dot(datas,w) # Nx1
    h = sigmoid(z)          # Nx1
    Error = labs-h          # Nx1
    w = w + alpha*np.dot(datas.T,Error)
    return w
```

进行训练，返回w

```python
def train_LR(datas,labs,n_epoch=2,alpha=0.005):
    N,D = np.shape(datas)
    w = np.ones([D,1])   # Dx1
    # 进行n_epoch轮迭代
    for i in range(n_epoch):
        w = weight_update(datas,labs,w,alpha)
        error_rate=test_accuracy(datas,labs,w)
        print("epoch %d error %.3f%%"%(i,error_rate*100))
    return w
```

```python
def test_accuracy(datas,labs,w):
    N,D = np.shape(datas)
    z = np.dot(datas,w) # Nx1
    h = sigmoid(z)          # Nx1
    lab_det = (h>0.5).astype(np.float)
    error_rate=np.sum(np.abs(labs-lab_det))/N
    return error_rate
```

测试程序

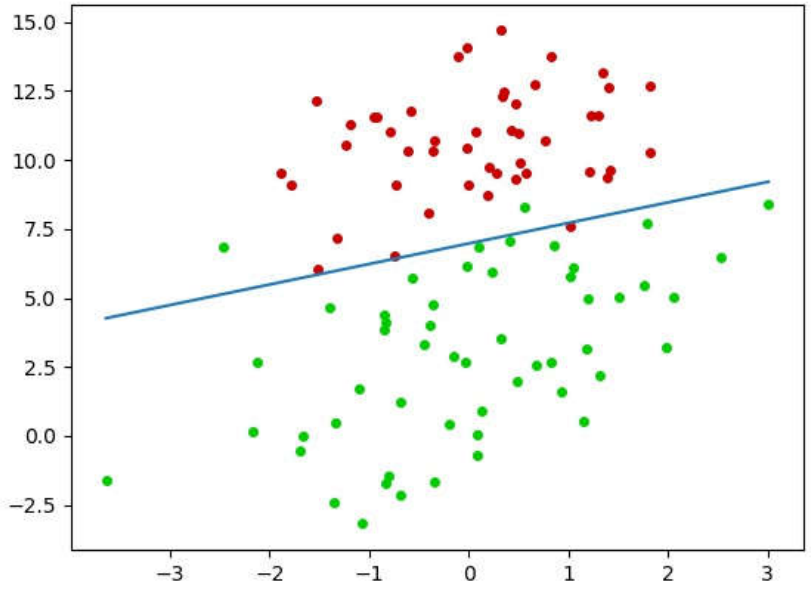
```
def load_dataset(file):
    with open(file,"r") as f:
        lines = f.read().splitlines()

    # 取 lab 维度为 N x 1
    labs = [line.split("\t")[-1] for line in lines]
    labs = np.array(labs).astype(np.float)
    labs= np.expand_dims(labs,axis=-1) # Nx1

    # 取数据 增加 一维全是1的特征
    datas = [line.split("\t")[:-1] for line in lines]
    datas = np.array(datas).astype(np.float)
    N,D = np.shape(datas)
    # 增加一个维度
    datas = np.c_[np.ones([N,1]),datas]
    return datas,labs
```

数据增加
全是1的维度

testset.txt | logisticRegression.py | PeakDensity.

1	-0.017612	14.053064	0
2	-1.395634	4.662541	1
3	-0.752157	6.538620	0
4	-1.322371	7.152853	0
5	0.423363	11.054677	0
6	0.406704	7.067335	1
7	0.667394	12.741452	0
8	-2.460150	6.866805	1
9	0.569411	9.548755	0
10	-0.026632	10.427743	0
11	0.850433	6.920334	1
12	1.347183	13.175500	0
13	1.176813	3.167020	1
14	-1.781871	9.097953	0
15	-0.566606	5.749003	1
16	0.931635	1.589505	1
17	-0.024205	6.151823	1
18	-0.036453	2.690988	1
19	-0.196949	0.444165	1
20	1.014459	5.754399	1
21	1.985298	3.230619	1
22	-1.693453	-0.557540	1
23	-0.576525	11.778922	0
24	-0.346811	-1.678730	1
25	-2.124484	2.672471	1
26	1.217916	9.597015	0

```
def draw_desion_line(datas,labs,w,name="0.jpg"):
    dic_colors={0:(.8,0,0),1:(0,.8,0)}

    # 画数据点
    for i in range(2):
        index = np.where(labs==i)[0]
        sub_datas = datas[index]
        plt.scatter(sub_datas[:,1],sub_datas[:,2],s=16.,color=dic_colors[i])

    # 画判决线
    min_x = np.min(datas[:,1])
    max_x = np.max(datas[:,1])
    w = w[:,0]
    x = np.arange(min_x,max_x,0.01)
    y = -(x*w[1]+w[0])/w[2]
    plt.plot(x,y)

    plt.savefig(name)


if __name__ == "__main__":
    ''' 实验1 基础测试数据'''
    # 加载数据
    file = "testset.txt"
    datas,labs = load_dataset(file)

    weights = train_LR(datas,labs,alpha=0.002,n_epoch=550)
    print(weights)
    draw_desion_line(datas,labs,weights)
```



随机梯度下降：

将数据分成小的**Batch，**每个Batch更新一次，可以增加训练的速度和稳定性

```
# 随机梯度下降
def train_LR_batch(datas,labs,batchsize,n_epoch=2,alpha=0.005):
    N,D = np.shape(datas)
    # weight 初始化
    w = np.ones([D,1])  # Dx1
    N_batch = N//batchsize
    for i in range(n_epoch):
        # 数据打乱
        rand_index  = np.random.permutation(N).tolist()
        # 每个batch 更新一下weight
        for j in range(N_batch):
            index = rand_index[j*batchsize:(j+1)*batchsize]
            batch_datas = datas[index]
            batch_labs = labs[index]
            w=weight_update(batch_datas,batch_labs,w,alpha)

        error = test_accuracy(datas,labs,w)
        print("epoch %d  error  %.2f%%"%(i,error*100))
    return w
```

```
'''实验2 马疝数据集'''
# 加载训练数据
train_file = "horse_train.txt"
train_datas,train_labs = load_dataset(train_file)

# 加载测试数据
test_file = "horse_test.txt"
test_datas,test_labs = load_dataset(test_file)

# 梯度下降
# weights = train_LR(train_datas,train_labs,alpha=0.001,n_epoch=90)
# print(weights)

# 随机梯度下降
weights = train_LR_batch(train_datas,train_labs,batchsize=2,n_epoch=50,alpha=0.001)
print(weights)

acc = test_accuracy(test_datas,test_labs,weights)
print(acc)

# 截取几个维度画图
index = [0,4,5]
sub_datas = train_datas[:,index]
sub_weights = weights[index]
draw_desion_line(sub_datas,train_labs,sub_weights,name="horse.jpg")
```