Softmax Regression

下面介绍一下指数族分布的另外一个例子。之前的逻辑回归中，可以用来解决二分类的问题。对于有k中可能结果的时候，问题就转化为多分类问题了，也就是接下来要说明的softmax regression问题。

1. 函数定义

在进行下一步推导前，我们先定义部分辅助函数。我们这里事先定义一个k-1维向量，这里具体对应于指数族分布的,具体定义如下:



我们定义表示的第一个元素，其他依次类推。

然后再引入一个函数，具体定义如下:



因此，我们可以轻易地得到:



注: 可以通过带入值的方式验证上式。

1. 模型推导

在softmax中,我们知道。我们设置的概率为,类似的有的概率为，并且我们轻易得到如下公式：





由于我们已经知道，综上可以得到:







我们对照指数族分布，其中我们已经定义，可以得到其他参数如下:







我们用来表示，目标是将我们的问题归整为一个变量的问题，从而更容易计算。对上面向量拆开计算得到，可以转化为。又由于我们已知，所以有，所以有。最后我们可以得到。由于指数组分布假设是关于输入的线性函数，所以得到在已经和的情况下的概率的公式，如下：



这里我们可以得到,如下:





注: 其中为取值的可能集合的大小，为一个的矩阵。

到这里我们可以定义我们的损失函数了，如下:



上式是一个向量，我们可以对这个向量继续求平方和，来衡量准确度，如下：



损失函数已经定义完成，我们就有了算法的截止条件了。接下来就是找到算法的最优迭代方向，也就是计算其偏导数了。

我们使用最大释然性来计算迭代方向。建模的原则是：对于一个已知的样本，我们需要找到一个方向去迭代，使得尽可能大，使得其他尽可能小。当然由于,所以我们只需要保证尽可能大即可。因此,在已经的m个样本的情况下，我们只需要保证下面的式子最大:



去自然对数后，我们定义最大似然函数:



当且仅当的时候, 。其他值为0，我们可以将连乘转化，如下：



然后对其求偏导数,对的时候后有：



对的时候，有：



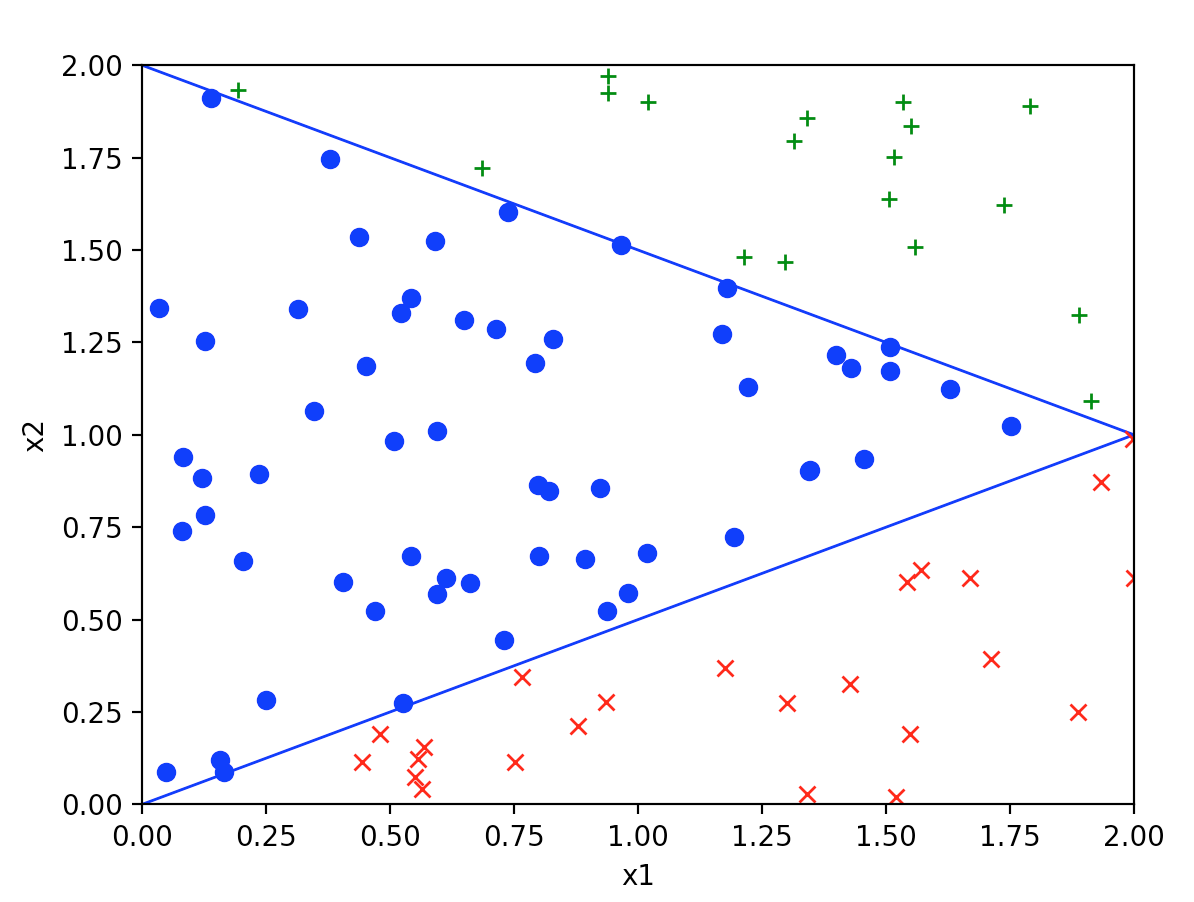
综上，可以得到:



注: 是一个向量，因为公式一次迭代更新一个维度下的一组。事实上，这里是一个向量求微分。如果我们对的某个元素进行微分，我们依然能够得到这个公式。

1. 解决实际问题

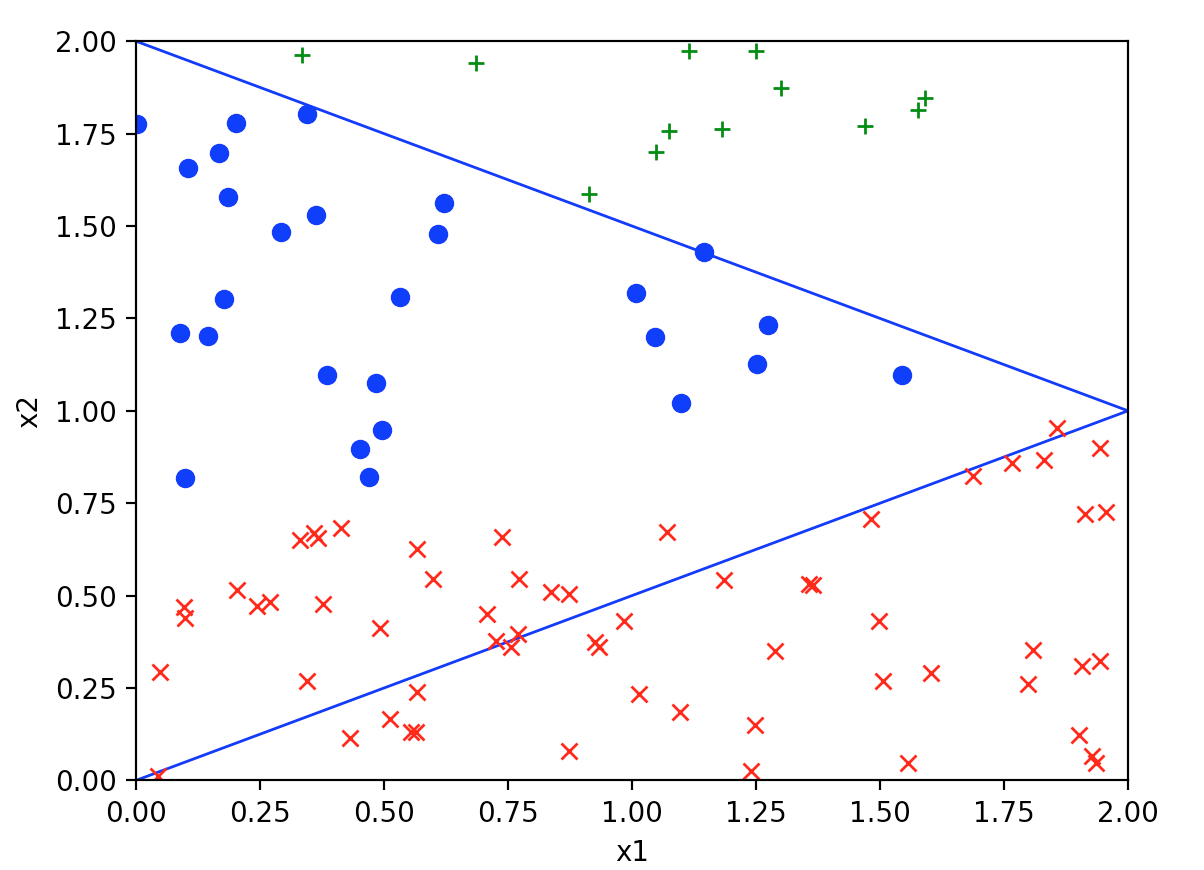
我们随机制造一组数据，具体的分类如下:



经过*33844次迭代之后，我们得到如下数值。*theta = [[17.418683742474045, 13.989348587756815, -37.731307869980014],[-33.56035031194091, 1.3797244978249763, 34.095843086377855],[19.97548491920147, -11.65484983882828, 7.628008093823735]]

注：迭代的次数越多数据会越准确，这里迭代了上三万次，实际上可以通过牛顿法来减少迭代的次数，这里就不再证明了。

然后，我们随机制造一组值，看看分类效果。



1. 代码
2. **import** random  
   **import** math  
   **import** numpy **as** np  
   **from** matplotlib **import** pyplot **as** plt  
   **from** matplotlib.lines **import** Line2D  
     
   g\_x\_arr=[[1, 0.035, 1.344], [1, 0.662, 0.598], [1, 1.791, 1.889], [1, 0.158, 0.12], [1, 1.55, 1.835], [1, 2.0, 0.613], [1, 1.176, 0.368], [1, 0.564, 0.043], [1, 1.559, 1.507], [1, 1.998, 0.988], [1, 0.082, 0.941], [1, 0.542, 1.371], [1, 0.542, 0.671], [1, 1.34, 1.856], [1, 0.049, 0.089], [1, 1.933, 0.871], [1, 1.753, 1.024], [1, 0.315, 1.341], [1, 0.829, 1.26], [1, 0.686, 1.721], [1, 1.222, 1.129], [1, 0.55, 0.075], [1, 0.767, 0.346], [1, 1.516, 1.752], [1, 1.347, 0.905], [1, 0.127, 0.782], [1, 1.169, 1.272], [1, 1.301, 0.273], [1, 0.081, 0.739], [1, 0.203, 0.658], [1, 0.347, 1.064], [1, 0.793, 1.193], [1, 1.428, 0.326], [1, 0.509, 0.983], [1, 0.12, 0.884], [1, 0.251, 0.282], [1, 0.73, 0.445], [1, 1.889, 1.323], [1, 1.314, 1.795], [1, 1.297, 1.467], [1, 1.669, 0.613], [1, 0.753, 0.114], [1, 0.94, 1.972], [1, 0.738, 1.603], [1, 1.508, 1.237], [1, 0.979, 0.572], [1, 0.128, 1.254], [1, 0.569, 0.155], [1, 0.88, 0.211], [1, 0.405, 0.603], [1, 1.02, 1.9], [1, 0.438, 1.535], [1, 1.506, 1.638], [1, 1.712, 0.394], [1, 0.556, 0.124], [1, 0.444, 0.115], [1, 0.595, 1.009], [1, 0.165, 0.089], [1, 1.57, 0.634], [1, 1.429, 1.181], [1, 0.8, 0.671], [1, 1.914, 1.091], [1, 0.594, 0.569], [1, 0.935, 0.277], [1, 0.47, 0.522], [1, 0.94, 1.924], [1, 0.194, 1.933], [1, 0.612, 0.613], [1, 0.236, 0.894], [1, 1.888, 0.251], [1, 1.548, 0.191], [1, 1.543, 0.603], [1, 1.521, 0.02], [1, 0.923, 0.856], [1, 0.649, 1.31], [1, 0.379, 1.746], [1, 1.345, 0.902], [1, 0.937, 0.524], [1, 1.018, 0.68], [1, 1.738, 1.623], [1, 1.534, 1.9], [1, 0.139, 1.911], [1, 1.508, 1.173], [1, 0.798, 0.865], [1, 0.451, 1.186], [1, 1.63, 1.123], [1, 0.82, 0.848], [1, 1.213, 1.48], [1, 0.894, 0.664], [1, 1.456, 0.934], [1, 0.59, 1.525], [1, 0.522, 1.329], [1, 1.179, 1.396], [1, 0.527, 0.273], [1, 1.399, 1.215], [1, 0.966, 1.514], [1, 1.341, 0.028], [1, 0.479, 0.191], [1, 1.193, 0.724], [1, 0.714, 1.285]]  
   g\_Ty\_arr=[[0, 0], [0, 0], [0, 1], [0, 0], [0, 1], [1, 0], [1, 0], [1, 0], [0, 1], [1, 0], [0, 0], [0, 0], [0, 0], [0, 1], [0, 0], [1, 0], [0, 0], [0, 0], [0, 0], [0, 1], [0, 0], [1, 0], [1, 0], [0, 1], [0, 0], [0, 0], [0, 0], [1, 0], [0, 0], [0, 0], [0, 0], [0, 0], [1, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 1], [0, 1], [0, 1], [1, 0], [1, 0], [0, 1], [0, 0], [0, 0], [0, 0], [0, 0], [1, 0], [1, 0], [0, 0], [0, 1], [0, 0], [0, 1], [1, 0], [1, 0], [1, 0], [0, 0], [0, 0], [1, 0], [0, 0], [0, 0], [0, 1], [0, 0], [1, 0], [0, 0], [0, 1], [0, 1], [0, 0], [0, 0], [1, 0], [1, 0], [1, 0], [1, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 1], [0, 1], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 1], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [1, 0], [1, 0], [0, 0], [0, 0]]  
   g\_y\_arr=[2, 2, 1, 2, 1, 0, 0, 0, 1, 0, 2, 2, 2, 1, 2, 0, 2, 2, 2, 1, 2, 0, 0, 1, 2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 1, 1, 1, 0, 0, 1, 2, 2, 2, 2, 0, 0, 2, 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 2, 1, 2, 0, 2, 1, 1, 2, 2, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2]  
     
   **def** phi(i,theta,x):  
    theta\_n = len(theta)  
    numerator = math.exp(np.dot(theta[i], x))  
    denominator = 0;  
    **for** j **in** range(theta\_n): *# maybe need to optimize* denominator = denominator + math.exp(np.dot(theta[j],x))  
    **return** numerator/denominator  
     
   **def** h\_theta(theta,x):  
    theta\_n = len(theta)  
    ret = []  
    **for** i **in** range(theta\_n):  
    ret.append(phi(i,theta,x))  
    **return** ret  
     
   **def** one(y,i):  
    **if** y == i:  
    **return** 1  
    **else**:  
    **return** 0  
     
   **def** derl(f,x\_arr,y\_arr,theta):  
    *# 0.  
    # f is the dimension to calc the partial derivative  
     
    # 1. calc the size  
    # theta\_n is the size of theta, equals to the length of Ty' result set -1 .Here, equals lenght of {0,1,2} = 3-1 =2  
    # m is the sum of samples  
    # x\_n is the dimension of the input variable 'x' + 1 (for x\_0 =1). Here is 2 + 1 = 3* theta\_n = len(theta)  
    m = len(x\_arr)  
    x\_n = len(x\_arr[0])  
    *# initial the output variable sum.Here is a vector, and the length of this vector is x\_n* sum = []  
    **for** x\_dim\_index **in** range(x\_n):  
    sum.append(0)  
     
    *# 2. calc the partial derivative* **for** i **in** range(x\_n):  
    sum[i] = 0  
    **for** j **in** range(m):  
    sum[i]=sum[i]+(one(y\_arr[j],f)-phi(f,theta,x\_arr[j]))\*x\_arr[j][i]  
    sum[i]=sum[i]/m  
    *#sum=plus\_vector(sum,theta[f],0.1)* **return** sum  
     
   **def** plus\_vector(arr1,arr2,a):  
    **return** [x + a \* y **for** x, y **in** zip(arr1, arr2)]  
     
   **def** update(theta,a,x\_arr,y\_arr):  
    theta\_n = len(theta)  
    **for** i **in** range(theta\_n):  
    theta[i]=plus\_vector(theta[i],derl(i, x\_arr,y\_arr,theta),a)  
    **return** theta  
     
   **def** judge1(theta,x\_arr,y\_arr\_vector,limit,debug):  
    j\_theta = J(theta,x\_arr,y\_arr\_vector)  
    **if** j\_theta < limit:  
    **return True  
    if** debug:  
    print(**"|J\_theta(x)| = "**, j\_theta,**"\n"**)  
    **return False  
     
   def** J(theta,x\_arr,y\_arr\_vector):  
    sum=0  
    m = len(x\_arr)  
    y\_len = len(y\_arr\_vector[0])  
    **for** i **in** range(m):  
    **for** j **in** range(y\_len):  
    sum = sum + (phi(j,theta,x\_arr[i]) - y\_arr\_vector[i][j])\*\*2  
    **return** sum  
     
   **def** calTheVaule():  
    *#theta=[[theta\_1\_1,theta\_1\_2],[theta\_2\_1,theta\_2\_2],[theta\_3\_1,theta\_3\_2]]* theta = [[1,1,1],[1,1,1],[1,1,1]]  
    a = 1  
    count = 0  
    **while** 1:  
    **if** judge1(theta,g\_x\_arr,g\_Ty\_arr,1,**True**):  
    **break** theta=update(theta,a,g\_x\_arr,g\_y\_arr)  
    count = count + 1  
    print(**"count="**,count,**"theta="**,theta)  
     
     
   **def** calcIndex(theta,x):  
    phi0 = phi(0,theta,x)  
    phi1 = phi(1,theta,x)  
    phi2 = phi(2,theta,x)  
    **if** phi0 >= phi1 **and** phi0 >=phi1:  
    **return** 0  
    **elif** phi1>= phi0 **and** phi1 >= phi2:  
    **return** 1  
    **else**:  
    **return** 2  
     
     
   **def** testValue():  
    *# limit =1， count = 33844* theta = [[17.418683742474045, 13.989348587756815, -37.731307869980014],[-33.56035031194091, 1.3797244978249763, 34.095843086377855],[19.97548491920147, -11.65484983882828, 7.628008093823735]]  
     
    samples = 100  
    figure, ax = plt.subplots()  
    *# 设置x，y值域* ax.set\_xlim(left=0, right=2)  
    ax.set\_ylim(bottom=0, top=2)  
    *# 两条line的数据* (line1\_xs, line1\_ys) = [(0, 2), (0, 1)]  
    (line2\_xs, line2\_ys) = [(0, 2), (2, 1)]  
    *# 创建两条线，并添加* ax.add\_line(Line2D(line1\_xs, line1\_ys, linewidth=1, color=**'blue'**))  
    ax.add\_line(Line2D(line2\_xs, line2\_ys, linewidth=1, color=**'blue'**))  
     
    **for** i **in** range(samples):  
    x\_0 = random.randint(0,2000)/1000  
    x\_1 = random.randint(0,2000)/1000  
    index = calcIndex(theta,[1,x\_0,x\_1])  
    **if** index == 0:  
    plt.plot(x\_0, x\_1, **'b--'**, marker=**'x'**, color=**'r'**)  
    **elif** index == 1:  
    plt.plot(x\_0, x\_1, **'b--'**, marker=**'+'**, color=**'g'**)  
    **else**:  
    plt.plot(x\_0, x\_1, **'b--'**, marker=**'o'**, color=**'b'**)  
    plt.xlabel(**"x1"**)  
    plt.ylabel(**"x2"**)  
    plt.plot()  
    plt.show()  
     
   **if** \_\_name\_\_== **"\_\_main\_\_"**:  
    *#calTheVaule()* testValue()