Classification

我们先考虑最简单的二分类的例子：如垃圾邮件分类、恶性肿瘤分类等，对于这类分类问题，数据的标签只有两个可能的值，0或1，一般用0表示negative的类别，用1表示positive的类别。

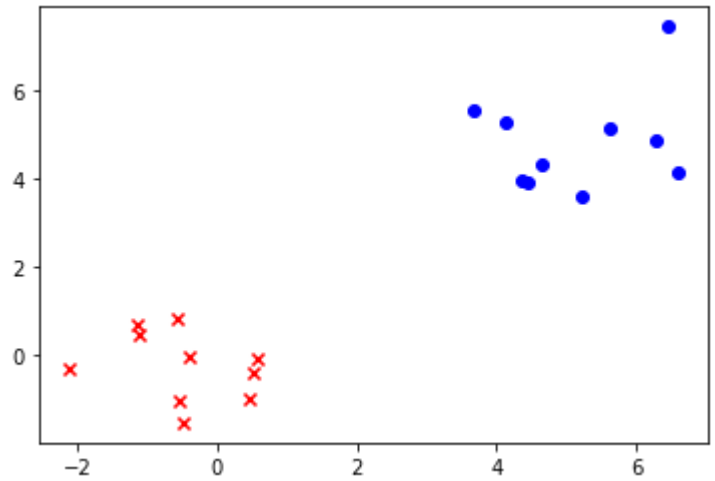
假设有以下的数据

In [1]:

```
# 导入包
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

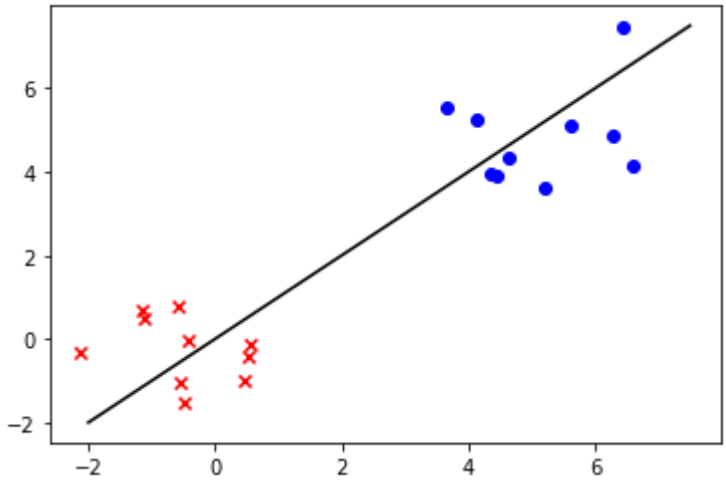
```
# 生成两堆数据
x1 = np.random.standard_normal(10)
y1 = np.random.standard_normal(10)

x2 = np.random.standard_normal(10)+5
y2 = np.random.standard_normal(10)+5

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.show()
```



我们尝试使用线性回归的方法来对这两堆数据进行分类，假设我们通过线性回归的方法得到了如下的曲线：
$h_\theta(x) = \theta^T x$

In [3]:

```
x = np.arange(-2, 8, 0.5)
y = x

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.plot(x, y, color='black')
plt.show()
```

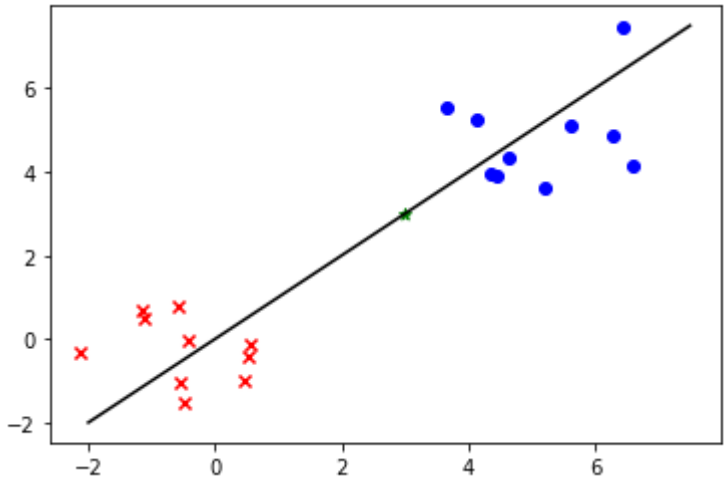


现在我们定义一个阈值，假设阈值为3，即$h_\theta(x) = 3$，对于一个新数据点：

- 如果$h_\theta(x_k) \geq 3$，我们认为是属于蓝色的点
- 而如果$h_\theta(x_k) < 3$，我们认为它是红色的点

In [4]:

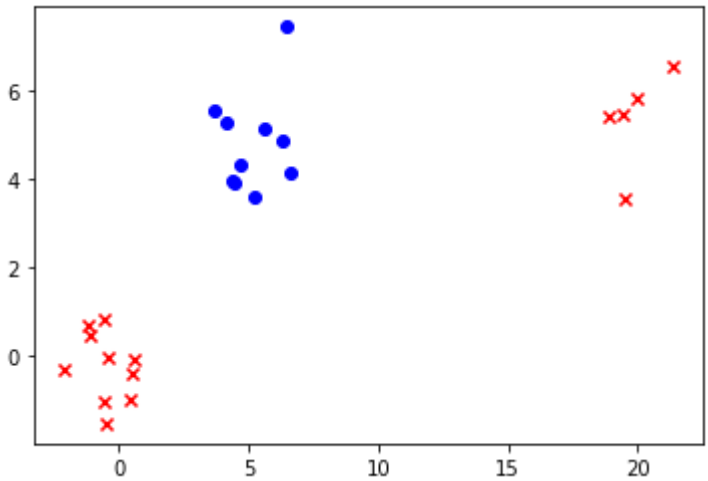
```
threshold_x = 3
threshold_y = 3

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.scatter(threshold_x, threshold_y, marker='*', color='green')

plt.plot(x, y, color='black')
plt.show()
```



目前来看可能是可以把这两对数据给分开，但假设现在数据集中加入了一些新的点：

In [5]:

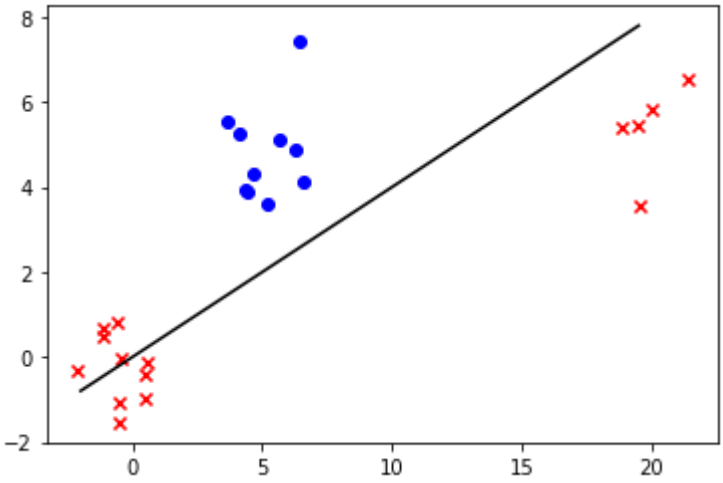
```
other_x = np.random.standard_normal(5)+20
other_y = np.random.standard_normal(5)+5

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.scatter(other_x, other_y, marker='x', color='red')

plt.show()
```



重新使用线性回归拟合后的结果可能是这样的：

In [6]:

```
x_1 = np.arange(-2, 20, 0.5)
y_1 = 0.4*x_1

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.scatter(other_x, other_y, marker='x', color='red')
plt.plot(x_1, y_1, color='black')

plt.show()
```



现在这种情况下我们无论怎么设置阈值都不能把红蓝两种数据分开了，所以我们也看出来，线性回归解决不了分类问题。

我们将使用Logistic Regression来解决分类问题，虽然它名字中是Regression，但它是分类算法

它把输出值限制在了0到1之间

hepothesis representation

在线性回归里，假设函数是$h_\theta(x) = \theta^T x$，对于LogisticRegression，我们会再此基础上将它作为sigmoid函数的输入，sigmoid函数又称为Logistic函数，也就是：

$$h_\theta(x) = g(\theta^T x)$$
$$g(z) = \frac{1}{1+e^{-z}}$$

In [1]:

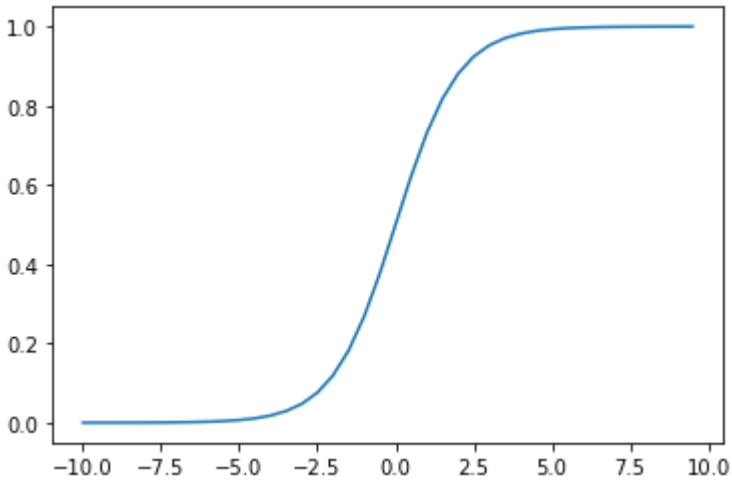
```
# 导入包
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
x = np.arange(-10, 10, 0.5)
y = 1/(1+np.exp(-x))

plt.plot(x, y)
```

Out[2]:

```
[<matplotlib.lines.Line2D at 0x15516d16630>]
```



我们之后会讨论如何找到合适的参数θ

现在我们解释假设函数输出的值：它的值表示了对于输入的x，它的标签为1的概率，也就是
$h_\theta(x) = P(y=1|x;\theta)$

而$P(y=1|x;\theta) + P(y=0|x;\theta) = 1$，$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$

在之前我们介绍了，对于LogisticRegression，它的假设函数是

$$h_\theta(x) = g(\theta^T x)$$
$$g(z) = \frac{1}{1+e^{-z}}$$

现在，我们同样定义一个阈值，当$h_\theta(x) \geq 0.5$时，预测它对应的标签y为1，当$h_\theta(x) < 0.5$时，预测它对应标签为0

我们注意到sigmoid函数当它的输入值大于0时，它的函数值大于0.5；当它的输入值小于0时，它的函数值小于0.5。

因此：

- 当$\theta^T x \geq 0$时，$h_\theta(x) \geq 0.5$，预测对应标签为1；
- 当$\theta^T x < 0$时，$h_\theta(x) < 0.5$，预测对应标签为0；

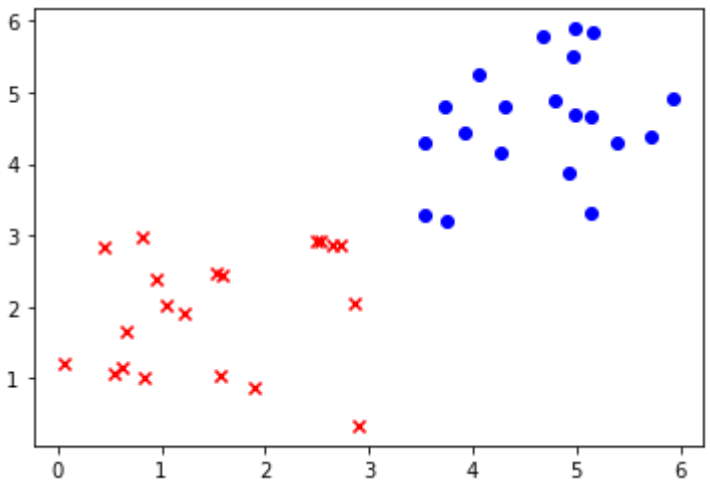
现在假设我们有如下的数据：（图中横轴为x_1，纵轴为x_2）

In [1]:

```
# 导入包
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
# 生成两堆数据
x1 = np.random.random(20)*3
y1 = np.random.random(20)*3

x2 = np.random.random(20)*3+3
y2 = np.random.random(20)*3+3

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.show()
```



我们的假设函数为$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$，我们还没有讨论怎么找到最优的参数$\theta$，但假设我们找到的

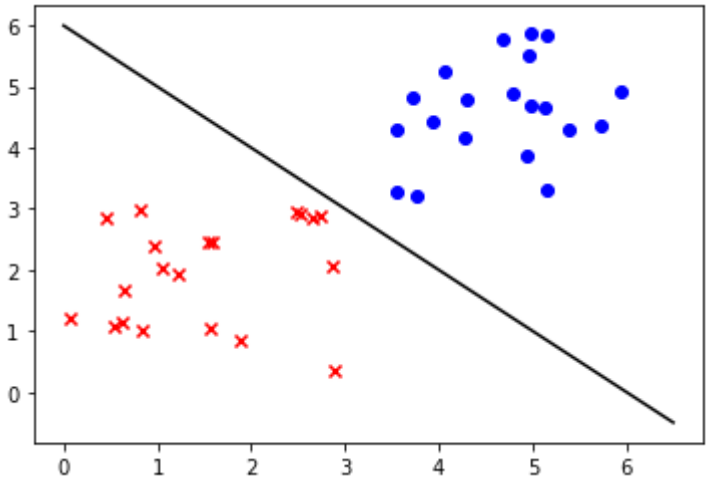
θ参数分别为$-6, 1, 1$，那么，$h_\theta(x) = g(-6 + x_1 + x_2)$。所以，按照我们设定的阈值：

- 当$-6 + x_1 + x_2 \geq 0$时，预测的标签为1；
- 当$-6 + x_1 + x_2 < 0$时，预测的标签为0；

我们可以画出这条直线：

In [3]:

```
x = np.arange(0, 7, 0.5)
y = 6-x

plt.scatter(x1, y1, marker='x', color='red')
plt.scatter(x2, y2, marker='o', color='blue')
plt.plot(x, y, color='black')
plt.show()
```



我们可以看出：

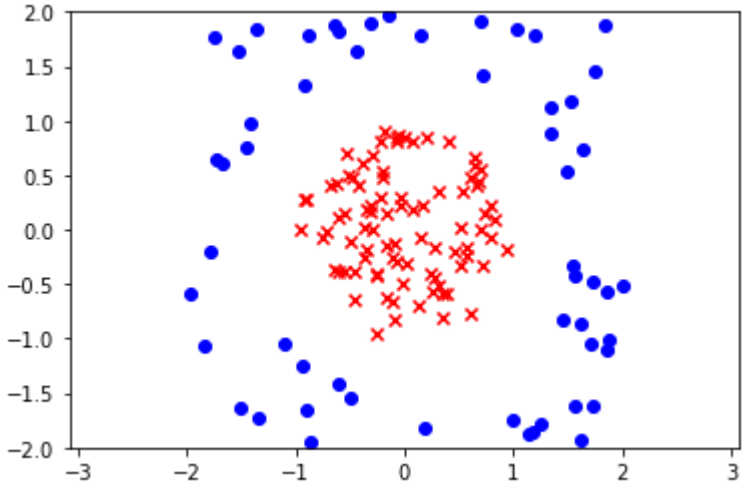
- 当$-6 + x_1 + x_2 \geq 0$时，对应曲线上方的区域，预测的标签为1；
- 当$-6 + x_1 + x_2 < 0$时，对应曲线下方的区域，预测的标签为0；

这个决策边界是由参数θ决定的

这个展示的例子是一个最简单的例子，可以举一个更复杂的例子，比如说，我们得到的数据如下图所示：

In [4]:

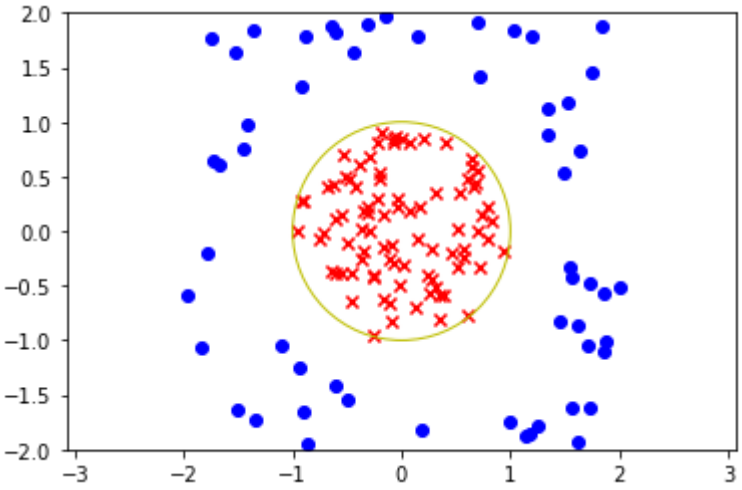
```
datax = np.random.random(100)*(-2)+1
datay = np.random.random(100)*(-2)+1

x3 = []
y3 = []
for i in range(100):
    if (np.sqrt(datax[i]**2 + datay[i]**2)) < 1:
        x3.append(datax[i])
        y3.append(datay[i])

datax = np.random.random(100)*(-4)+2
datay = np.random.random(100)*(-4)+2

x4 = []
y4 = []
for i in range(100):
    if (np.sqrt(datax[i]**2 + datay[i]**2)) > 1.5:
        x4.append(datax[i])
        y4.append(datay[i])

# x4_part1 = np.random.random(10)+1
# x4_part2 = np.random.random(10)*(-1)-1
# x4 = np.concatenate([x4_part1,x4_part2])

# y4_part1 = np.random.random(10)+1
# y4_part2 = np.random.random(10)*(-1)-1
# y4 = np.concatenate([y4_part1,y4_part2])

plt.scatter(x3, y3, marker='x', color='red')
plt.scatter(x4, y4, marker='o', color='blue')
plt.axis('equal')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.show()
```



如果我们的假设函数设置为$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$，找到的最优参数为$\theta$分别为-1，0，0，1，1，那么这个决策边界就是一个圆，能够很好的区分图中的数据。所以我们看出，如果有更高阶的多项式，我们就可以为更复杂的数据画出决策边界。

In [5]:

```
# 参数设置
x = 0  # 圆心的x轴坐标
y = 0  # 圆心的y轴坐标
r = 1  # 圆的半径

# 画圆
circle = plt.Circle((x, y), r, color='y', fill=False)
plt.gcf().gca().add_artist(circle)
plt.scatter(x3, y3, marker='x', color='red')
plt.scatter(x4, y4, marker='o', color='blue')
plt.axis('equal')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.show()

```



给定训练数据集：$\{(x^1, y^1), (x^2, y^2) \ldots, (x^m, y^m)\}$，$x \in R^{n+1}$，$y \in \{0, 1\}$，$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$，怎么找到最优的参数$\theta$

线性回归里使用的代价函数是MSE均方误差代价函数：

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

对于我们的Logistic Regression来说，$h_\theta(x)$它不是线性的，所以$J(\theta)$不是一个凸函数，使用梯度下降法对它很难进行优化，不容易得到最优解

我们定义：

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

我们可以将这个式子写成更简洁的形式：

$$Cost(h_\theta(x), y) = -(ylog(h_\theta(x)) + (1 - y)log(1 - h_\theta(x)))$$

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

这个更简洁的形式是由极大似然估计方法推导出来的：

因为数据的标签只有两种情况，0或1，假设：

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

对于第i个数据$\{x^i, y^i\}$，它在我们假设的分布中出现的概率就是$h_\theta(x^i)^{y^i}(1 - h_\theta(x^i)^{1-y^i}$，$y^i \in \{0, 1\}$

那么，利用极大似然估计的方法，似然函数为：

$$L(\theta) = \prod_{i=1}^{m} h_\theta(x^i)^{y^i}(1 - h_\theta(x^i)^{1-y^i})$$

$$logL(\theta) = log \prod_{i=1}^{m} h_\theta(x^i)^{y^i}(1 - h_\theta(x^i)^{1-y^i})$$

$$logL(\theta) = \sum_{i=1}^{m} logh_\theta(x^i)^{y^i}(1 - h_\theta(x^i)^{1-y^i})$$

$$logL(\theta) = \sum_{i=1}^{m} y^i logh_\theta(x^i) + (1 - y^i)log(1 - h_\theta(x^i))$$

$$\text{maxmize } logL(\theta) \text{ is equally minimize } -logL(\theta)$$

$$J(\theta) = -logL(\theta)$$

之后，我们可以通过梯度下降法来求解最优的θ：

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

而：

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

所以：

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

会发现，这个式子和线性回归的式子一样，不同之处在于，这里的$(h_\theta(x^i)$是一个非线性的函数了。

之前也介绍过特征缩放，特征缩放能帮助在梯度下降时收敛更快。