第二周学习笔记

2022-04-21

第一章　逻辑回归

1.1　分类问题

通过前面的学习我们知道监督学习分为“回归问题”和“分类问题”，前面的学习已经认识了什么是“回归问题”，接下来将学习“分类问题”的相关算法。

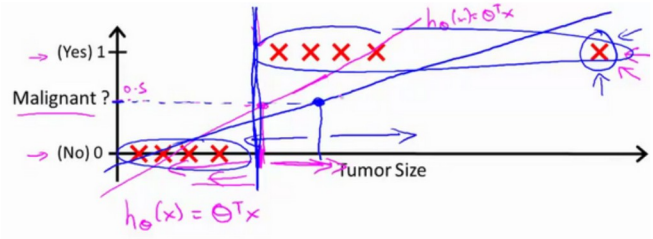
在分类问题中，要预测的变量 y 是离散的值，将学习一种叫做逻辑回归的算法，这是目前最流行使用最广泛的一种学习算法。

在分类问题中，我们尝试预测的是结果是否属于某一个类。例如判断一封电子邮件是否是垃圾邮件。

不妨用之前讲到的判断肿瘤的类别问题来认识分类问题。

从二元的分类问题开始讨论。

我们将因变量可能属于的两个类分别称为负向类和正向类，则因变量 $y \in (0,1)$, 其中 0 表示负向类，1 表示正向类。



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict “y = 1”

If $h_\theta(x) < 0.5$, predict “y = 0”

Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

如果我们用线性回归算法来解决一个分类问题，对于分类，y 取值为 0 或 1，但如果你使用的是线性回归，那么假设函数的输出值可能远大于 1，或者远小于 0，即使所

有训练样本的标签 y 都等于 0 或 1。尽管我们知道标签应该取值 0 或 1，但是如果算法得到的值远大于 1 或者远小于 0 的话，就会感觉很奇怪。

所以要研究的算法叫做逻辑回归算法，这个算法的性质是：它的输出值永远在 0 到 1 之间。

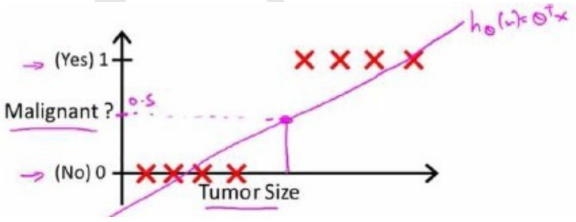
注意：逻辑回归算法是分类算法，我们将它作为分类算法使用。有时候可能因为这个算法的名字中出现了“回归”使你感到困惑，但逻辑回归算法实际上是一种分类算法，它适用于标签 y 取值离散的情况，如：1 0 0 1 0。

1.2　假说表示

首先提出问题：在分类问题中，要用什么样的函数来表示我们的假设？

此前讲过，希望分类输出的输出值在 0 和 1 之间，因此，我们希望想出一个满足某个性质的假设函数，这个性质是它的预测值要在 0 和 1 之间。

乳腺癌分类问题，我们可以用线性回归的方法求出适合数据的一条直线：

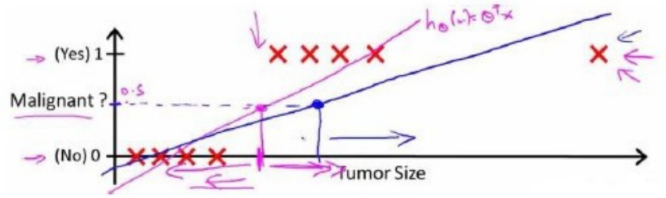


根据线性回归模型我们只能预测连续的值，然而对于分类问题，我们需要输出 0 或 1，我们可以预测：

当 $h_\theta(x) \geq 0.5$ 时，预测 $y=0$；

当 $h_\theta(x)<0.5$ 时，预测 $y=1$。

对于上图所示的数据，这样的一个线性模型似乎能很好地完成分类任务。假使我们又观测到一个非常大尺寸的恶性肿瘤，将其作为实例加入到我们的训练集中来，这将使得我们获得一条新的直线。



这时，再使用 0.5 作为阀值来预测肿瘤是良性还是恶性便不合适了。可以看出，线性回归模型，因为其预测的值可以超越 [0,1] 的范围，并不适合解决这样的问题。

引入一个新的模型，逻辑回归，该模型的输出变量范围始终在 0 和 1 之间。

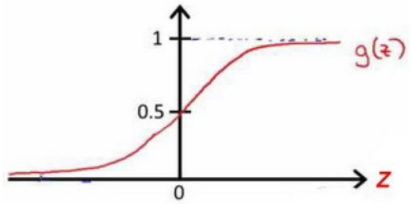
逻辑回归模型的假设是：

$$h_\theta(x) = g(\theta^T X)$$

其中：X 代表特征向量，g 代表逻辑函数，公式为：

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

该函数的图像为：



合起来，我们得到逻辑回归模型的假设：

对模型的理解：$g(z)=\frac{1}{1+e^{-z}}$。

$h_\theta(x)$ 的作用是，对于给定的输入变量，根据选择的参数计算输出变量 $=1$ 的可能性，即

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

例如，如果对于给定的，通过已经确定的参数计算得出，则表示有 70% 的几率为正向类，相应地为负向类的几率为 1-0.7=0.3。

1.3　判定边界

在逻辑回归中，我们预测：

当 $h_\theta(x) \geq 0.5$ 时，预测 $y=0$；当 $h_\theta(x)<0.5$ 时，预测 $y=1$。

根据上面绘制出的 S 形函数图像，我们知道当：

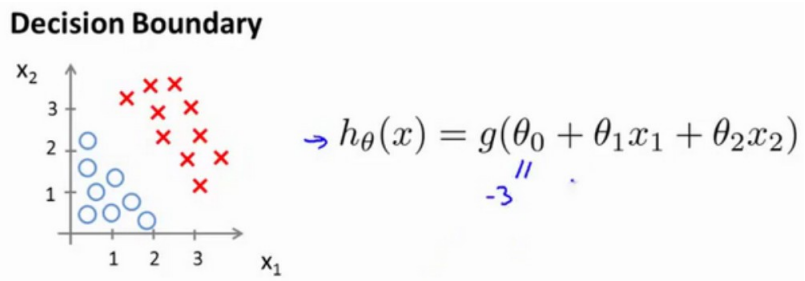
$$z = 0 \rightarrow g(z) = 0.5$$

$$z > 0 \rightarrow g(z) > 0.5$$

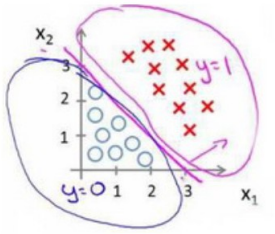
$$z < 0 \rightarrow g(z) < 0.5$$

有 $z = \theta^T x$，即 $\theta^T x \geq 0$，预测 y=1；$\theta^T x < 0$，预测 y=1。

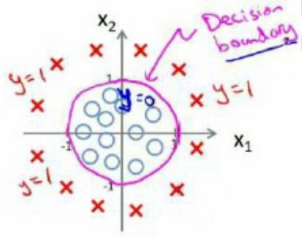
现在假设我们有一个模型：



并且参数 θ 是向量 [-3 1 1]。则当 $-3 + x_1 + x_2 \geq 0$，即 $x_1 + x_2 \geq 3$，模型将预测 y =1，我们可以绘制直线 $x_1 + x_2$ =3，这条线便是我们模型的分界线，将预测为 1 的区域和预测为 0 的区域分隔开。



假使我们的数据呈现这样的分布情况，怎样的模型才能适合呢？



因为需要用曲线才能分隔 y=1 的区域和 y=0 的区域，我们需要二次方特征。

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$ 是 [-1 0 0 1 1]，则我们得到的判定边界恰好是圆点在原点且半径为 1 的圆形。

我们可以用非常复杂的模型来适应非常复杂形状的判定边界。

1.4　代价函数

介绍如何拟合逻辑回归模型的参数 θ。具体来说，要定义用来拟合参数的优化目标或者叫代价函数，这便是监督学习问题中的逻辑回归模型的拟合问题。

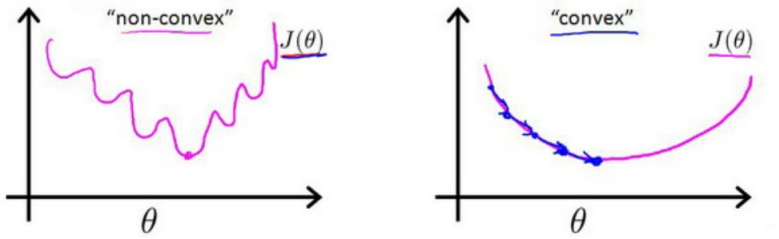
Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples　　$x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}$　　$x_0 = 1, y \in \{0, 1\}$

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

How to choose parameters θ ?

对于线性回归模型，我们定义的代价函数是所有模型误差的平方和。理论上来说，我们也可以对逻辑回归模型沿用这个定义，但是问题在于，当我们将 $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ 代入到这样定义了的代价函数中时，我们得到的代价函数将是一个非凸函数 $(non - convex function)$。



这意味着我们的代价函数有许多局部最小值，这将影响梯度下降算法寻找全局最小值。

线性回归的代价函数为：

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

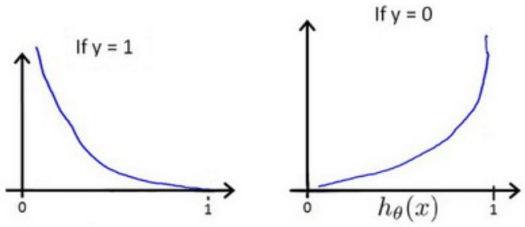
我们重新定义逻辑回归的代价函数为：

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

其中：

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

$h_\theta(x)$ 与 $Cost(h_\theta(x), y)$ 之间的关系如下图所示：



这样构建的 $Cost(h_\theta(x), y)$ 函数的特点是：

当实际的 y=1 且 $h_\theta(x)$ 也为 1 时误差为 0；

当 y=1 但 $h_\theta(x)$ 不为 1 时误差随着 $h_\theta(x)$ 变小而变大；

当实际的 y=0 且 $h_\theta(x)$ 也为 0 时代价为 0；

当 y=0 但 $h_\theta(x)$ 不为 0 时误差随着 $h_\theta(x)$ 的变大而变大。

将构建的 $Cost(h_\theta(x), y)$ 简化如下：

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

代入代价函数得到：

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

即：

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

Python 代码实现如下：

```
import numpy as np
def cost(theta, X, y):
  theta = np.matrix(theta)
  X = np.matrix(X)
  y = np.matrix(y)
  first = np.multiply(-y, np.log(sigmoid(X* theta.T)))
  second = np.multiply((1 - y), np.log(1 - sigmoid(X* theta.T)))
  return np.sum(first - second) / (len(X))
```

在得到这样一个代价函数以后，我们便可以用梯度下降算法来求得能使代价函数最小的参数了。算法为：

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

求导后得到：

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

代价函数 $J(\theta)$ 是一个凸函数，并且没有局部最优值。

推导过程：

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

考虑：

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

则：

$$\begin{aligned}
&y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-h_\theta(x^{(i)})) \\
&= y^{(i)}log(\frac{1}{1+e^{-\theta^T x^{(i)}}}) + (1-y^{(i)})log(1-\frac{1}{1+e^{-\theta^T x^{(i)}}}) \\
&= -y^{(i)}log(1+e^{-\theta^T x^{(i)}}) - (1-y^{(i)})log(1+e^{\theta^T x^{(i)}})
\end{aligned}$$

所以：

$$\begin{aligned}
\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}[-\frac{1}{m}\sum_{(i=1)}^{m}[-y^{(i)}log(1+e^{-\theta^T x^{(i)}}) - (1-y^{(i)})log(1+e^{\theta^T x^{(i)}})]] \\
&= -\frac{1}{m}\sum_{(i=1)}^{m}[-y^{(i)}\frac{-x_j^{(i)}e^{-\theta^T x^{(i)}}}{1+e^{-\theta^T x^{(i)}}} - (1-y^{(i)})\frac{x_j^{(i)}e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}} \\
&= -\frac{1}{m}\sum_{(i=1)}^{m}\frac{y^{(i)}x_j^{(i)} - x_j^{(i)}e^{\theta^T x^{(i)}} + y^{(i)}x_j^{(i)}e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}} \\
&= -\frac{1}{m}\sum_{(i=1)}^{m}\frac{y^{(i)}(1+e^{\theta^T x^{(i)}}) - e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}}x_j^{(i)} \\
&= -\frac{1}{m}\sum_{(i=1)}^{m}(y^{(i)} - \frac{e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}})x_j^{(i)} \\
&= -\frac{1}{m}\sum_{(i=1)}^{m}(y^{(i)} - \frac{1}{1+e^{-\theta^T x^{(i)}}})x_j^{(i)} \\
&= -\frac{1}{m}\sum_{(i=1)}^{m}[y^{(i)} - h_\theta(x^{(i)})]x_j^{(i)} \\
&= \frac{1}{m}\sum_{(i=1)}^{m}[-y^{(i)} + h_\theta(x^{(i)})]x_j^{(i)}
\end{aligned}$$

注：虽然得到的梯度下降算法表面上看上去与线性回归的梯度下降算法一样，但是这里的 $h_\theta(x) = g(\theta^T x)$ 与线性回归中不同，所以实际上是不一样的。另外，在运行梯度下降算法之前，进行特征缩放依旧是非常必要的。

一些梯度下降算法之外的选择：除了梯度下降算法以外，还有一些常被用来令代价函数最小的算法，这些算法更加复杂和优越，而且通常不需要人工选择学习率，通常比梯度下降算法要更加快速。这些算法有：共轭梯度，局部优化法，有限内存局部优化法。

1.5　简化的成本函数和梯度下降

找出一种稍微简单一点的方法来写代价函数，来替换上节用的方法。同时我们还要弄清楚如何运用梯度下降法，来拟合出逻辑回归的参数。

逻辑回归的代价函数：

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

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

Note: $y = 0$ or 1 always

这个式子可以合并成：

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

即，逻辑回归的代价函数：

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

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

根据这个代价函数，为了拟合出参数，该怎么做？

我们要试图找尽量让 $J(\theta)$ 取得最小值的参数 θ。

最小化代价函数的方法，是使用梯度下降法。

这是我们的代价函数：

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

如果我们要最小化这个关于 θ 的函数值，这就是我们通常用的梯度下降法的模板：

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

我们要反复更新每个参数，用这个式子来更新，就是用它自己减去学习率 α 乘以后面的微分项。

求导后得到：

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

如果你计算一下的话，你会得到这个等式：

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

这个式子正是我们用来做线性回归梯度下降的。

那么，线性回归和逻辑回归是同一个算法吗？

实际上，假设的定义发生了变化。

对于线性回归假设函数：

$$h_\theta(x) = \theta^T X = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

而现在逻辑函数假设函数：

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

因此，即使更新参数的规则看起来基本相同，但由于假设的定义发生了变化，所以逻辑函数的梯度下降，跟线性回归的梯度下降实际上是两个完全不同的东西。

在谈论线性回归的梯度下降法时，学习了如何监控梯度下降法以确保其收敛，通常也可以把同样的方法用在逻辑回归中，来监测梯度下降，以确保它正常收敛。

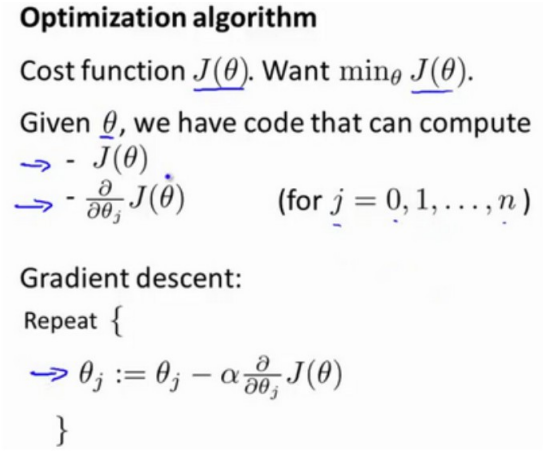
当使用梯度下降法来实现逻辑回归时，我们有这些不同的参数 θ，就是 $\theta_0, \theta_1, \theta_2$ 一直到 θ_n，我们需要用这个表达式来更新这些参数。我们还可以使用 for 循环来更新这些参数值，用 $i \in (1, n)$，或者 $i \in (1, n+1)$。当然，不用 for 循环也是可以的，理想情况下，我们更提倡使用向量化的实现，可以把所有这些 n 个参数同时更新。

我们之前在谈线性回归时讲到的特征缩放，我们看到了特征缩放是如何提高梯度下降的收敛速度的，这个特征缩放的方法，也适用于逻辑回归。如果你的特征范围差距很大的话，那么应用特征缩放的方法，同样也可以让逻辑回归中，梯度下降收敛更快。

1.6　高级优化

一些高级优化算法和一些高级的优化概念，利用这些方法，我们就能够使通过梯度下降，进行逻辑回归的速度大大提高，而这也将使算法更加适合解决大型的机器学习问题，比如，我们有数目庞大的特征量。

现在我们换个角度来看什么是梯度下降，我们有个代价函数 $J(\theta)$，而我们想要使其最小化，那么我们需要做的是编写代码，当输入参数 θ 时，它们会计算出两样东西：$J(\theta)$ 以及 J 等于 0、1 直到 n 时的偏导数项。



假设我们已经完成了可以实现这两件事的代码，那么梯度下降所做的就是反复执行这些更新。

另一种考虑梯度下降的思路是：我们需要写出代码来计算 $J(\theta)$ 和这些偏导数，然后把这些插入到梯度下降中，然后它就可以为我们最小化这个函数。

对于梯度下降来说，我认为从技术上讲，你实际并不需要编写代码来计算代价函数 $J(\theta)$。你只需要编写代码来计算导数项，但是，如果你希望代码还要能够监控这些 $J(\theta)$ 的收敛性，那么我们就需要自己编写代码来计算代价函数 $J(\theta)$ 和偏导数项 $\frac{\partial}{\partial\theta_j}J(\theta)$。

共轭梯度法 BFGS (变尺度法) 和 L-BFGS (限制变尺度法) 就是其中一些更高级的优化算法，它们需要有一种方法来计算 $J(\theta)$，以及需要一种方法计算导数项，然后使用比梯度下降更复杂的算法来最小化代价函数。

这三种算法有许多优点：一个是使用这其中任何一个算法，你通常不需要手动选择学习率 α，所以对于这些算法的一种思路是，给出计算导数项和代价函数的方法，你可以认为算法有一个智能的内部循环，而且，事实上，他们确实有一个智能的内部循环，称为线性搜索算法，它可以自动尝试不同的学习速率 α，并自动选择一个好的学习速率 α，因此它甚至可以为每次迭代选择不同的学习速率，那么你就不需要自己选择。

1.7　多类别分类：一对多

如何使用逻辑回归来解决多类别分类问题，具体来说，想通过一个叫做”一对多”的分类算法。

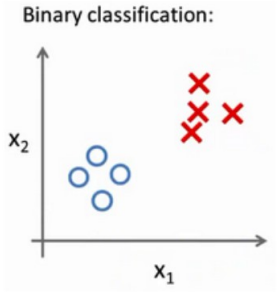
先看这样一些例子：

第一个例子：假如说你现在需要一个学习算法能自动地将邮件归类到不同的文件夹里，或者说可以自动地加上标签，那么，你也许需要一些不同的文件夹，或者不同的标签来完成这件事，来区分开来自工作的邮件、来自朋友的邮件、来自家人的邮件或者是有关兴趣爱好的邮件，那么，我们就有了这样一个分类问题：其类别有四个，分别用 $y=1, y=2, y=3, y=4$ 来代表。

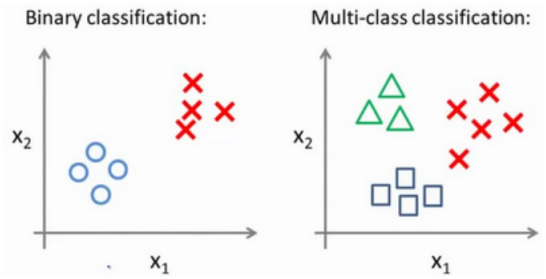
第二个例子是有关药物诊断的，如果一个病人因为鼻塞来到你的诊所，他可能并没有生病，用 $y=1$ 这个类别来代表；或者患了感冒，用 $y=2$ 来代表；或者得了流感用 $y=3$ 来代表。

第三个例子：如果你正在做有关天气的机器学习分类问题，那么你可能想要区分哪些天是晴天、多云、雨天、或者下雪天，对上述所有的例子，y 可以取一个很小的数值，一个相对”谨慎”的数值，比如 1 到 3、1 到 4 或者其它数值，以上说的都是多类分类问题，顺便一提的是，对于下标是 0 1 2 3，还是 1 2 3 4 都不重要，我更喜欢将分类从 1 开始标而不是 0，其实怎样标注都不会影响最后的结果。

然而对于之前的一个，二元分类问题，我们的数据看起来可能是像这样：

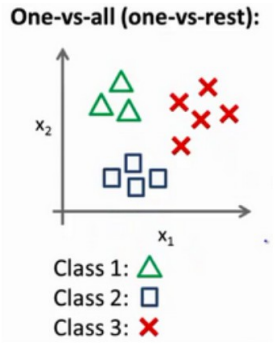


对于一个多类分类问题，我们的数据集或许看起来像这样：



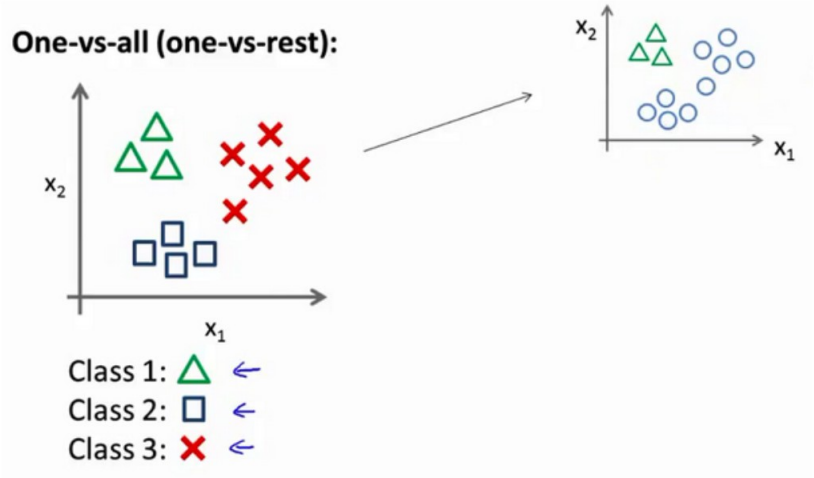
当我们使用 3 种不同的符号来代表 3 个类别，问题就是给出 3 个类型的数据集，如何得到一个学习算法来进行分类呢？

我们现在已经知道如何进行二元分类，可以使用逻辑回归，对于直线或许也知道，可以将数据集一分为二为正类和负类。用一对多的分类思想，我们可以将其用在多类分类问题上。



现在我们有一个训练集，好比上图表示的有 3 个类别，我们用三角形表示 $y=1$，方框表示 $y=2$，叉叉表示 $y=3$。我们下面要做的就是使用一个训练集，将其分成 3 个二元分类问题。

我们先从用三角形代表的类别 1 开始，实际上我们可以创建一个，新的”伪”训练集，类型 2 和类型 3 定为负类，类型 1 设定为正类，我们创建一个新的训练集，如下图所示的那样，我们要拟合出一个合适的分类器。



这里的三角形是正样本，而圆形代表负样本。可以这样想，设置三角形的值为 1，圆形的值为 0，下面我们来训练一个标准的逻辑回归分类器，这样我们就得到一个正边界。

为了能实现这样的转变，我们将多个类中的一个类标记为正向类 $(y=1)$，然后将其他所有类都标记为负向类，这个模型记作 $h_\theta^{(1)}(x)$。接着，类似地第我们选择另一个类标记为正向类 $(y=2)$，再将其它类都标记为负向类，将这个模型记作 $h_\theta^{(2)}(x)$。, 依此类推。最后我们得到一系列的模型简记为：$h_\theta^{(i)}(x)=p(y=i|x;\theta)$。其中：$i=(1,2,3,\cdots,k)$。

最后，在我们需要做预测时，我们将所有的分类机都运行一遍，然后对每一个输入变量，都选择最高可能性的输出变量。

总之，我们已经把要做的做完了，现在要做的就是训练这个逻辑回归分类器：$h_\theta^{(i)}(x)$，其中 i 对应每一个可能的 $y=i$，最后，为了做出预测，我们给出输入一个新的 x 值，用

这个做预测。

我们要做的就是在我们三个分类器里面输入 x，然后我们选择一个让 $h_\theta^{(i)}(x)$ 最大的 i，即 $max_i h_\theta^{(i)}(x)$。

第二章　正则化

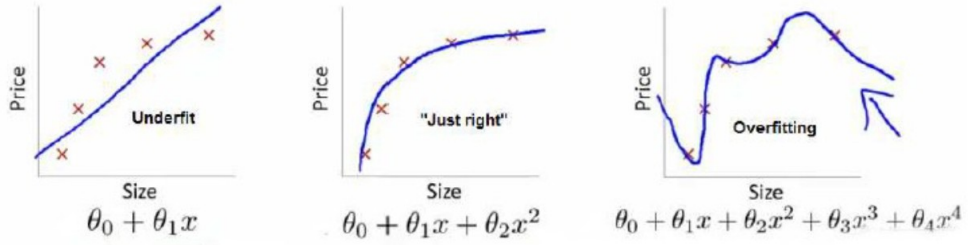
2.1　过拟合的问题

线性回归和逻辑回归，它们能够有效地解决许多问题，但是当将它们应用到某些特定的机器学习应用时，会遇到过拟合的问题，可能会导致它们效果很差。

正则化的技术，它可以改善或者减少过度拟合问题。

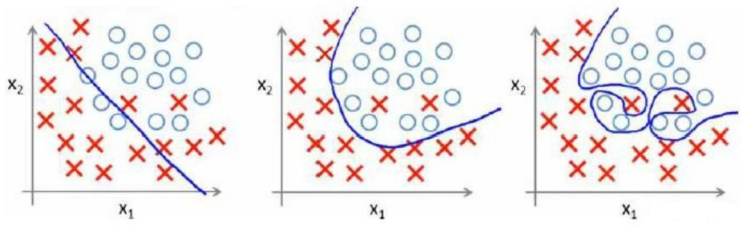
如果我们有非常多的特征，我们通过学习得到的假设可能能够非常好地适应训练集（代价函数可能几乎为 0），但是可能会不能推广到新的数据。

下图是一个回归问题的例子：



第一个模型是一个线性模型，欠拟合，不能很好地适应我们的训练集；第三个模型是一个四次方的模型，过于强调拟合原始数据，而丢失了算法的本质：预测新数据。我们可以看出，若给出一个新的值使之预测，它将表现的很差，是过拟合，虽然能非常好地适应我们的训练集但在新输入变量进行预测时可能会效果不好；而中间的模型似乎最合适。

分类问题中也存在这样的问题：



就以多项式理解，x 的次数越高，拟合的越好，但相应的预测的能力就可能变差。

问题是，如果我们发现了过拟合问题，应该如何处理？

1. 丢弃一些不能帮助我们正确预测的特征。可以是手工选择保留哪些特征，或者使用一些模型选择的算法来帮忙（例如 PCA）

2. 正则化。保留所有的特征，但是减少参数的大小（$magnitude$）

2.2　代价函数

上面的回归问题中如果我们的模型是：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

我们可以从之前的事例中看出，正是那些高次项导致了过拟合的产生，所以如果我们能让这些高次项的系数接近于 0 的话，我们就能很好的拟合了。

我们要做的就是在一定程度上减小这些参数 θ 的值，这就是正则化的基本方法。我们决定要减少 θ_3 和 θ_4 的大小，我们要做的便是修改代价函数，在其中 θ_3 和 θ_4 设置一点惩罚。这样做的话，我们在尝试最小化代价时也需要将这个惩罚纳入考虑中，并最终导致选择较小一些的 θ_3 和 θ_4。

修改后的代价函数如下：

$$min_\theta \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 10000\theta_4^2]$$

通过这样的代价函数选择出的 θ_3 和 θ_4 对预测结果的影响就比之前要小许多。假如我们有非常多的特征，我们并不知道其中哪些特征我们要惩罚，我们将对所有的特征进行惩罚，并且让代价函数最优化的软件来选择这些惩罚的程度。这样的结果是得到了一个较为简单的能防止过拟合问题的假设：

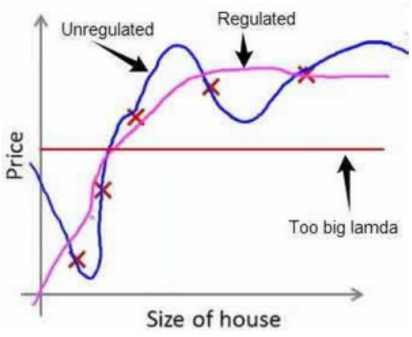
$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{i=1}^{n} \theta_j^2]$$

其中 λ 又称为正则化参数。

注：根据惯例，我们不对 θ_0 进行惩罚。经过正则化处理的模型与原模型的可能对比如下图所示：

如果选择的正则化参数 λ 过大，则会把所有的参数都最小化了，导致模型变成 $h_\theta(x) = \theta_0$，也就是上图中红色直线所示的情况，造成欠拟合。

那为什么增加的一项 $\lambda = \sum_{j=1}^{n} \theta_j^2$ 可以使 θ 的值减小呢？



因为如果我们令 λ 的值很大的话，为了使 $CostFunction$ 尽可能的小，所有的 θ 的值（不包括 θ_0）都会在一定程度上减小。

但若 λ 的值太大了，那么 θ（不包括 θ_0）都会趋近于 0，这样我们所得到的只能是一条平行于 x 轴的直线。所以对于正则化，我们要取一个合理的 λ 的值，这样才能更好的应用正则化。

2.3　正则化线性回归

对于线性回归的求解，我们之前推导了两种学习算法：一种基于梯度下降，一种基于正规方程。正则化线性回归的代价函数为：

$$J(\theta)=\frac{1}{2m}[((h_\theta(x^{(i)})-y^{(i)})^2+\lambda\sum_{j=1}^{n}\theta_j^2]$$

如果我们要使用梯度下降法令这个代价函数最小化，因为我们未对 θ_0 进行正则化，所以梯度下降算法将分两种情形：

$$\theta_0:=\theta_0-a\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)})-y^{(i)})x_0^{(i)})$$

$$\theta_j:=\theta_j-a[\frac{1}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}+\frac{\lambda}{m}\theta_j]$$

$$for j=1,2,\cdots,n$$

对上面的算法中 $j=1,2,\cdots,n$ 时的更新式子进行调整可得：

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

可以看出，正则化线性回归的梯度下降算法的变化在于，每次都在原有算法更新规则的基础上令 θ 值减少了一个额外的值。

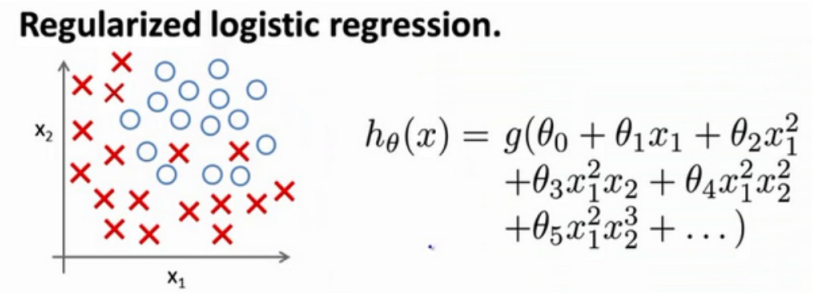
我们同样也可以利用正规方程来求解正则化线性回归模型，方法如下所示：

图中的矩阵尺寸为 $(n+1)*(n+1)$。

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

2.4　正则化的逻辑回归模型

针对逻辑回归问题，我们在之前的课程已经学习过两种优化算法：我们首先学习了使用梯度下降法来优化代价函数 $J(\theta)$，接下来学习了更高级的优化算法，这些高级优化算法需要你自己设计代价函数 $J(\theta)$。



自己计算导数同样对于逻辑回归，我们也给代价函数增加一个正则化的表达式，得到代价函数：

$$J(\theta) = \frac{1}{m} \sum_{(i=1)}^{m} [-y^{(i)} log(h_\theta(x^{(i)})) - (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Python 代码：

```
import numpy as np
def costReg(theta, X, y, learningRate):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    first = np.multiply(-y, np.log(sigmoid(X*theta.T)))
    second = np.multiply((1 - y), np.log(1 - sigmoid(X*theta.T)))
    reg = (learningRate / (2 * len(X))* np.sum(np.power(theta[:,1:theta.
                                        shape[1]],2))
    return np.sum(first - second) / (len(X)) + reg
```

要最小化该代价函数，通过求导，得出梯度下降算法为：

$$\theta_0 := \theta_0 - a \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)})$$

$$\theta_j := \theta_j - a [\frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j]$$

$$for j = 1, 2, \cdots, n$$

注：看上去同线性回归一样，但是知道 $h_\theta(x) = g(\theta^T X)$，所以与线性回归不同。

值得注意的是参数 θ_0 的更新规则与其他情况不同。

注意：

1. 虽然正则化的逻辑回归中的梯度下降和正则化的线性回归中的表达式看起来一样，但由于两者的 $h_\theta(x)$ 不同所以还是有很大差别。

2.θ_0 不参与其中的任何一个正则化。