

单变量线性回归

假设函数(hypothesis function)

$h_{\theta}(x) = \theta_0 + \theta_1 x$ 给定 θ_1, θ_2 有唯一的 $h(x)$, $h(x)$ 是关于 x 的函数, 用来预测 x 对应的 y 。

根据训练集 计算出 θ_0, θ_1 使得 $h(x)$ 和 y 差距最小。 用差值的平方衡量差距。

如何做到这一点呢?

代价函数/平方误差函数(cost function)

我们的目标是找出 θ_1, θ_2 使下面的代价函数最小。

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

J 是关于 θ_1, θ_2 的函数, 用来衡量 $h(x)$ 与样本数据的差距。对于不同的 θ 都有一个假设函数。

其实还有其他代价函数, 但上面这个代价函数是对大多数线性回归问题都合理的选择。

注: 这里的 $1/2m$ 中的2是为了使求导时消掉2, 也可以不用除以 $2m$, 这并不影响结果。为什么要求导呢? 因为要求最小值。

梯度下降(gradient descent)

过程:

1. 给 θ_1, θ_2 一个初始值
2. 不断改变 θ_1, θ_2 使 J 减小
3. 重复直到找到最小值

梯度下降法是用来求函数最小值的方法。

注：梯度的方向是方向导数取最大值的方向/函数增长速度最快的方向。其实就是找到 $\frac{\partial}{\partial \theta_j} J(\theta_j) = 0$ 时的参数值。

数学表达式：

repeat until convergence { $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ for $j=0; j \leq 1; j++$ }

- $:=$ 为赋值
- α 是学习率 (**learning rate**)，它决定了我们沿着能让代价函数下降程度最大的方向向下迈出的步子有多大。太大的话可能会错过最小值。
- $(\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1), \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1))$ 是在 (θ_0, θ_1) 的梯度

在梯度下降算法中，我们要更新 θ_0 和 θ_1 ，然后更新 $J(\theta_0)$ 和 $J(\theta_1)$ 。需要同时更新 θ_0 和 θ_1

所以正确算法如下：

$$a := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$b := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := a$$

$$\theta_1 := b$$

随着梯度下降法的运行，你移动的幅度会自动变得越来越小，直到最终移动幅度非常小，当到达最小值时，移动幅度为0，你会发现，已经收敛到局部极小值。梯度下降法只能找到局部最小值。

梯度下降的线性回归

对我们之前的线性回归问题运用梯度下降法，关键在于求出代价函数的导数，即：

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$j = 0 \text{ 时: } \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$j = 1 \text{ 时: } \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

则算法改写成：

Repeat {

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

$$\theta_1 := \theta_1 - a \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

}

我们使用的梯度下降算法也称为批量梯度下降，因为每次梯度下降都要遍历所有的训练样本（在计算微分求导项时，我们需要进行求和运算）。

实际上在数据量较大的情况下，梯度下降法比正规方程要更适用一些。

多变量线性回归

假设函数和代价函数

在单变量线性回归中， x 代表房屋面积， y 代表价格。如果影响价格的不止房屋面积一个因素呢？例如房间数楼层等。这样就有了多个变量的模型。模型中的特征为 (x_1, x_2, \dots, x_n) 。

多变量的假设 h 表示为： $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ ，为了使得公式能够简化一些，引入 $x_0 = 1$ ，则公式转化为： $h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

公式可以简化为： $h_{\theta}(x) = \theta^T X = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

相应的代价函数可写为 $J(\theta_0, \theta_1 \dots \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

上面 $X = (x_0, x_1, \dots, x_n)$ 是向量形式，现在把向量形式换成矩阵表达式：

$$X = \begin{pmatrix} x_0^0, x_0^1, \dots, x_0^n \\ x_1^0, x_1^1, \dots, x_1^n \\ \dots \\ x_m^0, x_m^1, \dots, x_m^n \end{pmatrix}$$

$$\text{则有 } J(\theta) = \frac{1}{2}(X\theta - y)^2 = \frac{1}{2}(X\theta - y)^T (X\theta - y)$$

注： X 为 m 行 n 列的矩阵（ m 为样本个数， n 为特征个数）， θ 为 m 行1列的矩阵， y 为 m 行1列的矩阵

$$h_{\theta}(x^{(i)}) = h_{\theta}(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_n x_n^{(i)}$$

多变量梯度下降

我们的目标和单变量线性回归问题中一样，是要找出使得代价函数最小的一系列参数。

数学表达式：

repeat until convergence { $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n)$ for $j=0; j \leq n; j++$ }

把 $h_{\theta}(x)$ 带入并求导得到： $\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$ for $j=0; j \leq n; j++$

当 $n \geq 1$ 时， $\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$
$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

特征缩放

以房价问题为例，假设我们使用两个特征，房屋的尺寸和房间的数量，尺寸的值为 0-2000 平方英尺，而房间数量的值则是 0-5，以两个参数分别为横纵坐标，绘制代价函数的等高线图能，看出图像会显得很扁，梯度下降算法需要非常多次的迭代才能收敛。

解决的方法是尝试将所有特征的尺度都尽量缩放到-1到1之间。

最简单的方法是令： $x_n = \frac{x_n - \mu_n}{s_n}$ ，其中 μ_n 是平均值， s_n 是标准差。

debugging 和学习率

可以通过绘制 $J(\theta)$ 曲线来观察曲线在经过多少次迭代之后收敛。

梯度下降算法的每次迭代受到学习率的影响，如果学习率 α 过小，则达到收敛所需的迭代次数会非常高；如果学习率 α 过大，每次迭代可能不会减小代价函数，可能会越过局部最小值导致无法收敛。

通常可以考虑尝试些学习率：

$\alpha = 0.01, 0.03, 0.1, 0.3, 1, 3, 10$

多项式回归

线性回归并不适用于所有数据，有时我们需要曲线来适应我们的数据，比如一个二次方模型： $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$ 或者三次方模型： $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$

根据上面说过的多变量线性回归的 $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ ，我们可以令 $x_2 = x_1^2, x_3 = x_1^3$ 。再用线性回归的方法，拟合一个多次函数。

注：如果我们采用多项式回归模型，在运行梯度下降算法前，特征缩放非常有必要。

根据对数据的形状的了解，通过选择不同的特征，可以得到更好的模型。

正规方程


梯度下降是经过多次迭代，得到代价函数的最小值的方法。而正规方程法只需要一步就可以得到最小值，而不需要经过多次迭代。

根据高数知识可以知道， $\frac{\partial}{\partial \theta_j} J(\theta_j) = 0$ 时，可求得极值点。这里 θ 是一个多维向量，在多元线性回归中他是 $n+1$ 维的。

代价函数： $J(\theta_0, \theta_1 \dots \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$ 对于每个 j ，求满足 $\frac{\partial}{\partial \theta_j} J(\theta_j) = 0$ 条件的 $\theta_0, \dots, \theta_n$

看下面的例子：

Examples: $m = 4$.

	Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
 x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

注：增加 x_0 这一列，是为了计算方便，让 θ 和 x 一一对应，可以看上面。

$$J(\theta) = \frac{1}{2} (X\theta - y)^T (X\theta - y)$$

$$= \frac{1}{2} (\theta^T X^T - y^T) (X\theta - y)$$

$$= \frac{1}{2} (\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta - y^T y)$$

接下来对 $J(\theta)$ 偏导，需要用到以下几个矩阵的求导法则：

$$\frac{dAB}{dB} = A^T$$

$$\frac{dX^T AX}{dX} = 2AX$$

所以有：

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{2} (2X^T X \theta - X^T y - (y^T X)^T - 0)$$

$$= \frac{1}{2} (2X^T X \theta - X^T y - X^T y - 0)$$

$$= X^T X \theta - X^T y$$

$$\text{令 } \frac{\partial J(\theta)}{\partial \theta} = 0,$$

$$\text{则有 } \theta = (X^T X)^{-1} X^T y$$

梯度下降	正规方程
需要选择学习率 α	不需要
需要多次迭代	一次运算得出
当特征数量 n 大时也能较好适用	需要计算 $(X^T X)^{-1}$ 如果特征数量 n 较大则运算代价大，因为矩阵逆的计算时间复杂度为 $O(n^3)$ ，通常来说当 n 小于10000 时还是可以接受的
适用于各种类型的模型	只适用于线性模型，不适合逻辑回归模型等其他模型

总结一下，只要特征变量的数目并不大，标准方程是一个很好的计算参数 θ 的替代方法。具体地说，只要特征变量数量小于一万，我通常使用标准方程法，而不使用梯度下降法。

python实现线性回归

特征缩放

```
1 def normalize_feature(df):
2     # 特征缩放公式:  $x_n = (x_n - u_n) / s_n$ 
3     # .apply对每个元素进行以下运算, 如果axis=0, 那么当执行.mean()这类函数时,
4     # 会以列为单位
5     df.apply(lambda col: (col - col.mean()) / col.std, axis=0)
```

计算代价函数

```
1 # 计算代价函数
2 def ComputeCost(theta, X, y):
3     m = X.shape[0]
4     inner = np.dot(X, theta) - y
5     squire_sum = np.dot(inner.T, inner)
6     return squire_sum / (2 * m)
```

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$j = 0 \text{ 时: } \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$j = 1 \text{ 时: } \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

$$\text{实际上 } \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} X^T (X\theta - y)$$

计算梯度

```
1 # 计算梯度, J(theta0, theta1)对thetaj求偏导
2 def ComputeGradient(theta, X, y):
3     m = X.shape[0]
4     inner = X.T @ (X @ theta - y)
5     return inner / m
```


梯度下降算法

```
1  # 批量梯度下降
2  def batch_gradient_decent(theta, X, y, epoch, alpha=0.01):
3      #epoch 为迭代次数
4      # 初始化只有一个cost, 不停的梯度下降cost值在更新
5      cost = [ComputeCost(theta, X, y)]
6      # 拷贝一份, 不修改原来的theta
7      _theta = theta.copy
8      for _ in range(epoch):
9          _theta = _theta - alpha * ComputeGradient(_theta, X, y)
10         cost.append(ComputeCost(theta, X, y))
11     return _theta, cost
```

代价函数数据可视化

```
1  # 展示代价函数变化情况
2  plt.plot(np.arange(epoch+1), final_cost)
3  plt.show()
```

显示预测图像

```
1  b = final_theta[0] # intercept, y轴上的截距
2  m = final_theta[1] # slope, 斜率
3
4  plt.scatter(data.Population, data.Profit, label="Training data")
5  plt.plot(data.Population, data.Population*m + b,
6           label="Prediction")
7  # 设置图例
8  plt.legend(loc=2)
9  plt.show()
```