Week 3 Lecture Notes

机器学习：逻辑回归 - ML:Logistic Regression

现在我们正在从回归问题转向**分类问题**。不要被“逻辑回归”这个名字搞混了；它之所以这样命名，是因为历史的原因，实际上是一种处理分类问题的方法，而不是回归问题。

二分类问题 - Binary Classification

现在输出值y是只包含0或1，而不是一个连续的值。

y∈{0,1}

0通常被认为是“负类”，1为“正类”，但你自决定怎么表示。

我们现在只有两个类，叫做“二分类问题”。

一种方法是使用线性回归，并将所有大于0.5的预测映射为1，所有小于0.5的预测作为0。这种方法不能很好地工作，因为分类实际上不是一个线性函数。

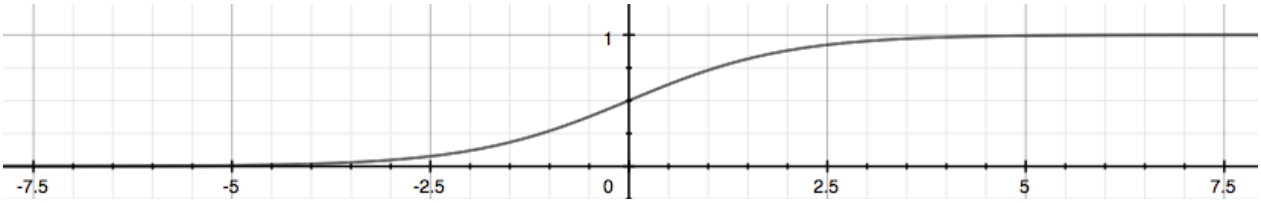
假设函数的表示

我们的假设应该满足：

$0 \leq h_\theta(x) \leq 1$

假设函数的新形式叫“Sigmoid 函数”，也称为“逻辑函数”：

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



这里所示的函数g(z)将任何实数映射到(0,1)区间，从而有助于将任意值函数转换为更适合分类的函数。试着玩一下sigmoid函数的交互图像：（https://www.desmos.com/calculator/bgontvxotm）。

我们从我们的旧的假设（线性回归）开始，通过将$\theta^T x$插入Logistic函数，限制输出范围为0到1。

h_θ表示给出我们的输出值（分类）为1的**概率**。例如，$h_\theta(x) = 0.7$ 表示有70%的概率，输出值（分类）是1。

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

输出值是0的概率，就是1减去输出值为1的概率（例如，如果输出值为1的概率是70%，那么输出值为0的概率是30%）。

决策边界 - Decision Boundary

为了得到离散的0或1分类，我们可以将假设函数的输出翻译如下：

$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(x) < 0.5 \rightarrow y = 0$$

逻辑函数g的性质是，当其输入大于或等于0时，其输出大于或等于0.5：

$$g(z) \geq 0.5$$
$$when\ z \geq 0$$

记住：

$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \rightarrow \infty, e^{-\infty} \rightarrow 0 \Rightarrow g(z) = 1$$
$$z \rightarrow -\infty, e^{\infty} \rightarrow \infty \Rightarrow g(z) = 0$$

因此，如果我们对G的输入是$\theta^T X$，那么这意味着：

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$

从这些性质可以看出：

$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

决策边界就是是将y＝0的区域和y＝1的区域分开的线。它是由我们的假设函数决定的。

例如：

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$
$$y = 1\ if\ 5 + (-1)x_1 + 0x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

在这种情况下，我们的决策边界是$x_1 = 5$代表的直线，其左边表示y=1，而右边表示y=0。

同样，对sigmoid函数g(z)(例如$\theta^T X$)的输入不一定是线性的，并且可以是圆的方程(例如，$z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$)或任何形状，只要函数适配数据。

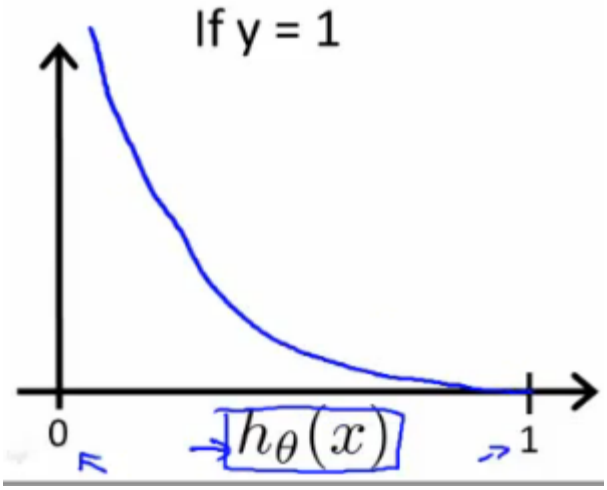
代价函数 - Cost Function

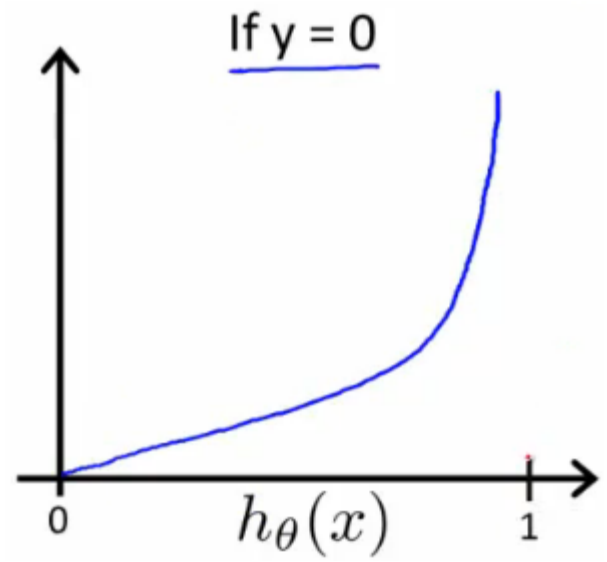
我们不能使用与线性回归相同的代价函数，因为Logistic函数会导致输出波动，产生许多局部最优。换句话说，它不是凸函数。

相反，logistic回归的代价函数看起来像：

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

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$
$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$





假设越远离实际的y值，代价函数输出越大。如果假设等于y，那代价就是0：

$$\text{Cost}(h_\theta(x), y) = 0 \text{ if } h_\theta(x) = y$$
$$\text{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 0 \text{ and } h_\theta(x) \to 1$$
$$\text{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 1 \text{ and } h_\theta(x) \to 0$$

如果我们的实际值“y”是0，如果假设函数也输出0，则代价函数将是0。如果我们的假设接近1，那么代价函数将接近无穷大。

如果我们的实际值“y”是1，如果假设函数也输出1，则代价函数将是0。如果我们的假设接近0，那么代价函数将接近无穷大。

注意，这种代价函数保证了逻辑回归中的$J(\theta)$是凸函数。

代价函数的简化与梯度下降 - Simplified Cost Function and Gradient Descent

我们可以将由两个条件组成的代价函数压缩成一个式子：

$$\text{Cost}(h_\theta(x), y) = -y\ \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

注意，当y等于1时，第二项$(1-y)\log(1-h_\theta(x))$将为零，不会影响结果。如果y等于0，则第一项$-y\log(h_\theta(x))$将为零，也不会影响结果。

那么整个代价函数如下：

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

矢量化的实现是：

$$h = g(X\theta)$$
$$J(\theta) = \frac{1}{m}\cdot\left(-y^T\log(h) - (1-y)^T\log(1-h)\right)$$

Gradient Descent - 梯度下降

记住梯度下降的一般形式是：

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

我们可以利用微积分求出导数部分：

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

请注意，该算法与我们在线性回归中使用的算法相同。我们仍然必须同时更新θ中的所有值。

矢量化的实现是：

$$\theta := \theta - \frac{\alpha}{m}X^T(g(X\theta) - \vec{y})$$

J(θ)的偏导数 - Partial derivative of J(θ)

首先计算sigmoid函数的导数（求J(θ)的偏导数时有用）：

$$\sigma(x)' = \left(\frac{1}{1+e^{-x}}\right)' = \frac{-(1+e^{-x})'}{(1+e^{-x})^2} = \frac{-1' - (e^{-x})'}{(1+e^{-x})^2} = \frac{0-(-x)'(e^{-x})}{(1+e^{-x})^2} = \frac{-(-1)(e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2}$$
$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) = \sigma(x)\left(\frac{+1-1+e^{-x}}{1+e^{-x}}\right) = \sigma(x)\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$$

现在我们计算偏导数：

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

矢量化的实现是：

$\nabla J(\theta)=\frac{1}{m}\cdot X^T\cdot(g\left(X\cdot\theta\right)-\vec{y})$

进阶优化 - Advanced Optimization

“共轭梯度”、“BFGS”和“L-BFGS”是更为复杂、更快的优化θ的方法，可以用来代替梯度下降。吴恩达教授建议不要自己编写这些更复杂的算法（除非您是数值计算专家），而是使用库，因为它们已经经过测试并高度优化。Octave内置这些函数。

我们首先需要提供一个函数，对于每一个θ，它计算以下两个值：

$J(\theta)$
$\frac{\partial}{\partial\theta_j}J(\theta)$

我们可以编写一个同时返回两个值的函数：

```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

然后我们可以使用Octave的“fminunc()”优化算法，通过“optimset()”函数创建一个包含要发送到“fminunc()”的选项的对象。（注意：MaxIter的值应该是整数，而不是视频中7:30时间的字符串）。

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
      initialTheta = zeros(2,1);
      [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
options);
```

我们向函数“fminunc()”提供我们的代价函数，θ的初始值向量，以及我们之前创建的“options”对象。

多分类问题：一对多 - Multiclass Classification: One-vs-all

现在我们将把数据分为许多类。而不是y= {0,1}，我们将扩展我们的定义，使得y= {0,1...n}。

在这种情况下，我们将问题划分为n+1(+1是因为索引是从0开始的)个二分类问题；在每个二分类问题中，我们预测“y”是其中一个类的概率。

$$
\begin{aligned}
&y \in \{0, 1 \dots n\} \\
&h_\theta^{(0)}(x) = P(y = 0 | x; \theta) \\
&h_\theta^{(1)}(x) = P(y = 1 | x; \theta) \\
&\cdots \\
&h_\theta^{(n)}(x) = P(y = n | x; \theta) \\
&\text{prediction} = \max_i (h_\theta^{(i)}(x))
\end{aligned}
$$

我们首先选择了一个类，然后把所有其他类集中成为第二类。重复这样做，对每种情况都是用二分类逻辑回归，然后使用返回假设的最大值对应的类别，作为预测。

机器学习：正则化 - ML:Regularization

过拟合问题 - The Problem of Overfitting

正则化是为了解决过拟合的问题。

高偏差或欠拟合发生在当我们假设函数h对数据的拟合程度不佳时。通常是由函数太简单或特征太少引起的。如果我们采用$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$作为假设函数，那么我们首先假设了线性模型可以很好地拟合训练数据，并且能够进行泛化，但实际有可能不会这样。

再看看另一种极端情况，**过拟合或高方差**。它是由假设函数适配数据过度引起的，它不能很好地泛化以预测新数据。这通常是由于假设函数过于复杂，使它产生了许多与数据无关的不必要的曲线和变化。

这个术语同时适用于线性回归和逻辑回归。有两个主要的方法来解决过拟合的问题：

- 减少特征数量：
  - 手动选择要保留的特征。
  - 使用模型选择算法（在本课程后面进行研究）。
- 正则化

保留所有的特征，但减少参数θ_j。

当我们有很多用处比较小的的特征时，正则化效果很好。

代价函数 - Cost Function

如果我们的假设函数过拟合，我们可以通过增加它们的代价来减少函数中的一些项的权重。

比如我们想使下面的函数更接近二次函数：

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

我们要消除$\theta_3 x^3$和$\theta_4 x^4$的影响。如果不能直接去掉这些特征，或者改变我们假设函数的形式，那就可以修改代价函数：

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

我们在最后增加了两个项款来增加θ_3 和 θ_4的成本。现在，算法为了使成本函数接近于零，会将θ_3和θ_4的值减少到接近零。这将极大地降低我们假设函数中的θ_3 和 θ_4的比重。

我们也可以在一个求和中正则化所有的θ参数：

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

λ或者叫lambda，是**正则化参数**。它决定了我们的θ参数的代价是多少。你可以在这个链接中直观的查看正则化的效果：https://www.desmos.com/calculator/1hexc8ntqp

利用上述包含额外求和项的代价函数，我们可以让假设函数的输出变得平滑，以减少过拟合。如果λ被选得太大，它可能会平滑得过多，导致欠拟合。

线性回归的正则化 - Regularized Linear Regression

我们可以将正则化应用于线性回归和逻辑回归。我们先进行线性回归。

梯度下降 - Gradient Descent

修改梯度下降函数以将θ_0与其他参数分开，因为我们不想惩罚θ_0。

$$
\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\
&\quad \theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad j \in \{1, 2 \ldots n\} \\
&\}
\end{aligned}
$$

$\frac{\lambda}{m}\theta_j$项就是正则化。

我们的更新规则也可以表示为：

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

上述方程中的第一项，$1 - \alpha\frac{\lambda}{m}$总是小于1。直观地说，你可以看到它在每一次更新中都减少了θ_j 的值。

注意，第二项和以前完全一样。

正规方程 - Normal Equation

现在我们在非迭代的正规方程的使用正则化。

除了在括号内加上另一个项之外，方程与原来的相同：

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

L是一个矩阵，在左上角是0，在对角线上都是1，在其他地方都是0。它的维数是(n+1)×(n+1)。直观地说，这就是一个单位矩阵（虽然不包括x_0），再乘以一个实数λ。

回想一下，如果m ≤ n，则$X^T X$是不可逆的。然而，当我们加上λ·L这个项时，$X^T X$ + λ·L就可逆了。

逻辑回归的正则化 - Regularized Logistic Regression

我们可以用类似于线性回归的方法来调整逻辑回归。让我们从代价函数开始。

代价函数 - Cost Function

回忆一下逻辑回归的成本函数是：

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

可以增加一个项来对这个等式进行正则化：

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

注：第二个求和，$\sum_{j=1}^{n} \theta_j^2$**明确地排除**了偏置项$\theta_0$。也就是说，θ向量从0到n的（包含n+1个值，$\theta_0$ through θ_n），而这个求和明确地跳过θ_0，仅仅计算从1到n。

梯度下降 - Gradient Descent

就像线性回归一样，我们希望**单独**更新θ_0和其他参数，因为我们不希望正则化θ_0。

$$\text{Repeat } \{$$

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

$$\theta_j := \theta_j - \alpha \ \left[\left(\frac{1}{m} \ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad j \in \{1, 2 \ldots n\}$$

$$\}$$

这与线性回归的梯度下降函数是相同的。

初始一个特征向量 - Initial Ones Feature Vector

常量特征 - Constant Feature

事实证明，在开始任何机器训练之前，在你的特征池中添加一个常量特征向量是至关重要的。通常，这个特征只是你训练数据中的一组。

具体地说，如果X是你的特征矩阵，那么X_0是一个所有值都为1的向量。

下面是一些关于使用这个常量特征向量的原因。第一部分从电气工程概念中得出一些类比，第二部分通过一个简单的机器学习例子来理解常量特征向量。

电气工程 - Electrical Engineering

From electrical engineering, in particular signal processing, this can be explained as DC and AC.

The initial feature vector X without the constant term captures the dynamics of your model. That means those features particularly record changes in your output y - in other words changing some feature X_i where $i \neq 0$ will have a change on the output y. AC is normally made out of many components or harmonics; hence we also have many features (yet we have one DC term).

The constant feature represents the DC component. In control engineering this can also be the steady state.

Interestingly removing the DC term is easily done by differentiating your signal - or simply taking a difference between consecutive points of a discrete signal (it should be noted that at this point the analogy is implying time-based signals - so this will also make sense for machine learning application with a time basis - e.g. forecasting stock exchange trends).

Another interesting note: if you were to play and AC+DC signal as well as an AC only signal where both AC components are the same then they would sound exactly the same. That is because we only hear changes in signals and Δ(AC+DC)=Δ(AC).

房价示例 - Housing price example

Suppose you design a machine which predicts the price of a house based on some features. In this case what does the ones vector help with?

Let's assume a simple model which has features that are directly proportional to the expected price i.e. if feature Xi increases so the expected price y will also increase. So as an example we could have two features: namely the size of the house in [m2], and the number of rooms.

When you train your machine you will start by prepending a ones vector X_0. You may then find after training that the weight for your initial feature of ones is some value θ0. As it turns, when applying your hypothesis function $h_\theta(X)$ - in the case of the initial feature you will just be multiplying by a constant (most probably θ0 if you not applying any other functions such as sigmoids). This constant (let's say it's θ_0 for argument's sake) is the DC term. It is a constant that doesn't change.

But what does it mean for this example? Well, let's suppose that someone knows that you have a working model for housing prices. It turns out that for this example, if they ask you how much money they can expect if they sell the house you can say that they need at least θ0 dollars (or rands) before you even use your learning machine. As with the above analogy, your constant θ0 is somewhat of a steady state where all your inputs are zeros. Concretely, this is the price of a house with no rooms which takes up no space.

However this explanation has some holes because if you have some features which decrease the price e.g. age, then the DC term may not be an absolute minimum of the price. This is because the age may make the price go even lower.

Theoretically if you were to train a machine without a ones vector $f_{AC}(X)$, it's output may not match the output of a machine which had a ones vector $f_{DC}(X)$. However, $f_{AC}(X)$ may have exactly the same trend as $f_{DC}(X)$ i.e. if you were to plot both machine's output you would find that they may look exactly the same except that it seems one output has just been shifted (by a constant). With reference to the housing price problem: suppose you make predictions on two houses house_AhouseA and

house_BhouseB using both machines. It turns out while the outputs from the two machines would different, the difference between houseA and houseB's predictions according to both machines could be exactly the same. Realistically, that means a machine trained without the ones vector f_AC could actually be very useful if you have just one benchmark point. This is because you can find out the missing constant by simply taking a difference between the machine's prediction an actual price - then when making predictions you simply add that constant to what even output you get. That is: if $house_{benchmark}$ is your benchmark then the DC component is simply $price(house_{benchmark}) - f_{AC}(features(house_{benchmark}))$

A more simple and crude way of putting it is that the DC component of your model represents the inherent bias of the model. The other features then cause tension in order to move away from that bias position.

Kholofelo Moyaba

更简单的方法 - A simpler approach

A "bias" feature is simply a way to move the "best fit" learned vector to better fit the data. For example, consider a learning problem with a single feature X_1. The formula without the X_0 feature is just $theta_1 * X_1 = y$. This is graphed as a line that always passes through the origin, with slope y/theta. The x_0 term allows the line to pass through a different point on the y axis. This will almost always give a better fit. Not all best fit lines go through the origin (0,0) right?

Joe Cotton