线性回归的概念

1、线性回归的原理

2、线性回归损失函数、代价函数、目标函数

3、优化方法(梯度下降法、牛顿法、拟牛顿法等)

4、线性回归的评估指标

5、sklearn参数详解

1、线性回归的原理

线性回归的一般形式：

有数据集$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$,其中,$x_i = (x_{i1}; x_{i2}; x_{i3}; \ldots; x_{id}), y_i \in R$
其中n表示变量的数量，d表示每个变量的维度。
可以用以下函数来描述y和x之间的关系：

$$\begin{aligned} f(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d \\ &= \sum_{i=0}^{d} \theta_i x_i \end{aligned}$$

如何来确定θ的值，使得$f(x)$尽可能接近y的值呢？均方误差是回归中常用的性能度量，即：

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

我们可以选择θ，试图让均方误差最小化：

极大似然估计（概率角度的诠释）

（找出与样本分布最接近的概率模型）

下面我们用极大似然估计，*来解释为什么要用均方误差作为性能度量*

我们可以把目标值和变量写成如下等式：

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

ϵ表示我们未观测到的变量的印象，即随机噪音。我们假定ϵ是独立同分布，服从高斯分布。（根据中心极限定理）

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

因此，

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

我们建立极大似然函数，即描述数据遵从当前样本分布的概率分布函数。由于样本的数据集独立同分布，因此可以写成

$$L(\theta) = p(\vec{y}|X;\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

选择θ，使得似然函数最大化，这就是极大似然估计的思想。

为了方便计算，我们计算时通常对对数似然函数求最大值：

$$\begin{aligned} l(\theta) = logL(\theta) &= log\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^{n} log\frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= nlog\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2}\sum_{i=1}^{n}((y^{(i)} - \theta^T x^{(i)})^2 \end{aligned}$$

显然，最大化$l(\theta)$即最小化 $\frac{1}{2}\sum_{i=1}^{n}((y^{(i)} - \theta^T x^{(i)})^2$。

这一结果即均方误差，因此用这个值作为代价函数来优化模型在统计学的角度是合理的。

2、线性回归损失函数、代价函数、目标函数

- 损失函数(Loss Function)：是定义在单个样本上的，算的是一个样本的误差。损失函数值越小，模型就越好。
- 代价函数(Cost Function)：是定义在整个训练集上的，是所有样本误差的平均，也就是损失函数的平均。
- 目标函数(Object Function)：代价函数和正则化函数，最终要优化的函数。

常用的损失函数包括：0-1损失函数、平方损失函数、绝对损失函数、对数损失函数等；常用的代价函数包括均方误差、均方根误差、平均绝对误差等。

思考题：既然代价函数已经可以度量样本集的平均误差，为什么还要设定目标函数?

回答：

当模型复杂度增加时，有可能对训练集可以模拟的很好，但是预测测试集的效果不好，出现过拟合现象，这就出现了所谓的“结构化风险”。结构风险最小化即为了防止过拟合而提出来的策略，定义模型复杂度为$J(F)$，目标函数可表示为：

$$\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda J(F)$$

例如有以上6个房价和面积关系的数据点，可以看到，当设定$f(x) = \sum_{j=0}^{5} \theta_j x_j$时，可以完美拟合训练集数据，但是，真实情况下房价和面积不可能是这样的关系，出现了过拟合现象。当训练集本身存在噪声时，拟合曲线对未知影响因素的拟合往往不是最好的。 通常，随着模型复杂度的增加，训练误差会减少；但测试误差会先增加后减小。我们的最终目的是使测试误差达到最小，这就是我们为什么需要选取适合的目标函数的原因。

- J(f)函数专门用来度量模型的复杂度，在机器学习中也叫正则化(regularization)。常用的有 L1，L2 范数。
- 结构风险小：模型简单；经验风险小：对历史数据拟合效果好

3、线性回归的优化方法

1、梯度下降法

设定初始参数θ,不断迭代，使得$J(\theta)$最小化(α为步长, f 为预测值, J(theta) 为损失函数)：

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

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^{n} (f_\theta(x)^{(i)} - y^{(i)})^2 \\
&= 2 * \frac{1}{2} \sum_{i=1}^{n} (f_\theta(x)^{(i)} - y^{(i)}) * \frac{\partial}{\partial \theta_j} (f_\theta(x)^{(i)} - y^{(i)}) \\
&= \sum_{i=1}^{n} (f_\theta(x)^{(i)} - y^{(i)}) * \frac{\partial}{\partial \theta_j} (\sum_{j=0}^{d} \theta_j x_j^{(i)} - y^{(i)}) \\
&= \sum_{i=1}^{n} (f_\theta(x)^{(i)} - y^{(i)}) x_j^{(i)}
\end{aligned}$$

即：

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{n} (y^{(i)} - f_\theta(x)^{(i)}) x_j^{(i)}$$

注：下标j表示第j个参数，上标i表示第i个数据点。

将所有的参数以向量形式表示，可得：

$$\theta = \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - f_\theta(x)^{(i)}) x^{(i)}$$

由于这个方法中，参数在每一个数据点上同时进行了移动，因此称为**「批梯度下降法」**，对应的，我们可以每一次让参数只针对一个数据点进行移动，即：

$$\theta = \theta + \alpha(y^{(i)} - f_\theta(x)^{(i)})x^{(i)}$$

这个算法成为随机梯度下降法，随机梯度下降法的好处是，当数据点很多时，运行效率更高；缺点是，因为每次只针对一个样本更新参数，未必找到最快路径达到最优值，甚至有时候会出现参数在最小值附近徘徊而不是立即收敛。但当数据量很大的时候，随机梯度下降法经常优于批梯度下降法。

当J为凸函数时，梯度下降法相当于让参数θ不断向J的最小值位置移动

梯度下降法的缺陷：如果函数为非凸函数，有可能找到的并非全局最优值，而是局部最优值。

2、最小二乘法矩阵求解

令

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \cdots \\ (x^{(n)})^T \end{bmatrix}$$

其中，

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \cdots \\ x_d^{(i)} \end{bmatrix}$$

由于

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(n)} \end{bmatrix}$$

$h_\theta(x)$可以写作

$$h_\theta(x) = X\theta$$

对于向量来说，有

$$z^T z = \sum_i z_i^2$$

因此可以把损失函数写作

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

为最小化$J(\theta)$,对θ求导可得:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta}\frac{1}{2}(X\theta - Y)^T(X\theta - Y) \\ &= \frac{1}{2}\frac{\partial}{\partial \theta}(\theta^T X^T X\theta - Y^T X\theta - \theta^T X^T Y - Y^T Y)\end{aligned}$$

中间两项互为转置，由于求得的值是个标量，矩阵与转置相同，因此可以写成

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

令偏导数等于零，由于最后一项和θ无关，偏导数为0。

因此，

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

利用矩阵求导性质，

$$\frac{\partial \vec{x}^T \alpha}{\partial \vec{x}} = \alpha$$

和

$$\frac{\partial A^T B}{\partial \vec{x}} = \frac{\partial A^T}{\partial \vec{x}} B + \frac{\partial B^T}{\partial \vec{x}} A$$

$$\begin{aligned}\frac{\partial}{\partial \theta}\theta^T X^T X\theta &= \frac{\partial}{\partial \theta}(X\theta)^T X\theta \\ &= \frac{\partial (X\theta)^T}{\partial \theta} X\theta + \frac{\partial (X\theta)^T}{\partial \theta} X\theta \\ &= 2X^T X\theta\end{aligned}$$

$$\frac{\partial J(\theta)}{\partial \theta} = X^T X\theta - X^T Y$$

令导数等于零，

$$X^T X\theta = X^T Y$$

$$\theta = (X^T X)^{(-1)} X^T Y$$

注：CS229视频中吴恩达的推导利用了矩阵迹的性质，可自行参考学习。

3、牛顿法

通过图例可知(参考吴恩达CS229),

$$f(\theta)' = \frac{f(\theta)}{\Delta}, \Delta = \theta_0 - \theta_1$$

$$\text{可求得，} \theta_1 = \theta_0 - \frac{f(\theta_0)}{f(\theta_0)'}$$

重复迭代，可以让逼近取到$f(\theta)$的最小值

当我们对损失函数$l(\theta)$进行优化的时候，实际上是想要取到$l'(\theta)$的最小值，因此迭代公式为：

$$\theta := \theta - \frac{l'(\theta)}{l''(\theta)}$$

$$\text{当}\theta\text{是向量值的时候，} \theta := \theta - H^{-1}\Delta_\theta l(\theta)$$

其中，$\Delta_\theta l(\theta)$是$l(\theta)$对θ_i的偏导数，H是$J(\theta)$的海森矩阵，

$$H_{ij} = \frac{\partial^2 l(\theta)}{\partial\theta_i \partial\theta_j}$$

问题：请用泰勒展开法推导牛顿法公式。

Answer：将$f(x)$用泰勒公式展开到第二阶，

$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2$

对上式求导，并令导数等于0，求得x值

$$f'(x) = f'(x_0) + f''(x_0)x - f''(x_0)x_0 = 0$$

可以求得，

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

牛顿法的收敛速度非常快，但海森矩阵的计算较为复杂，尤其当参数的维度很多时，会耗费大量计算成本。我们可以用其他矩阵替代海森矩阵，用拟牛顿法进行估计，

4、拟牛顿法

拟牛顿法的思路是用一个矩阵替代计算复杂的海森矩阵H，因此要找到符合H性质的矩阵。

要求得海森矩阵符合的条件，同样对泰勒公式求导$f'(x) = f'(x_0) + f''(x_0)x - f''(x_0)x_0$

令$x = x_1$，即迭代后的值，代入可得：

$$f'(x_1) = f'(x_0) + f''(x_0)x_1 - f''(x_0)x_0$$

更一般的，

$$f'(x_{k+1}) = f'(x_k) + f''(x_k)x_{k+1} - f''(x_k)x_k$$

$$f'(x_{k+1}) - f'(x_k) = f''(x_k)(x_{k+1} - x_k) = H(x_{k+1} - x_k)$$

x_k为第k个迭代值

即找到矩阵G，使得它符合上式。 常用的拟牛顿法的算法包括DFP，BFGS等，作为选学内容，有兴趣者可自行查询材料学习。

4、线性回归的评价指标

均方误差(MSE):$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$

均方根误差(RMSE)：$\sqrt{MSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}$

平均绝对误差(MAE)：$\frac{1}{m}\sum_{i=1}^{m}|(y^{(i)} - \hat{y}^{(i)}|$

但以上评价指标都无法消除*量纲不一致* 而导致的误差值差别大的问题，最常用的指标是R^2,可以避免量纲不一致问题

$$R^2 := 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(\bar{y} - \hat{y}^{(i)})^2} = 1 - \frac{\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\frac{1}{m}\sum_{i=1}^{m}(\bar{y} - \hat{y}^{(i)})^2} = 1 - \frac{MSE}{VAR}$$

我们可以把R^2理解为，回归模型可以成功解释的数据方差部分在数据固有方差中所占的**比例**，R^2越接近1，表示可解释力度越大，模型拟合的效果越好。

5、sklearn.linear_model参数详解：

fit_intercept : 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距

normalize: 默认为false. 当fit_intercept设置为false的时候，这个参数会被自动忽略。如果为True,回归器会标准化输入参数：减去平均值，并且除以相应的二范数。当然啦，在这里还是建议将标准化的工作放在训练模型之前。通过设置sklearn.preprocessing.StandardScaler来实现，而在此处设置为false

copy_X : 默认为True, 否则X会被改写

n_jobs: int 默认为1. 当-1时默认使用全部CPUs ??(这个参数有待尝试)

可用属性：

coef_:训练后的输入端模型系数，如果label有两个，即y值有两列。那么是一个2D的array

intercept_: 截距

可用的methods:

fit(X,y,sample_weight=None): X: array, 稀疏矩阵 [n_samples,n_features] y: array [n_samples, n_targets] sample_weight: 权重 array [n_samples] 在版本0.17后添加了sample_weight

get_params(deep=True)： 返回对regressor 的设置值

predict(X): 预测 基于 R^2值

score： 评估

参考https://blog.csdn.net/weixin_39175124/article/details/79465558 (https://blog.csdn.net/weixin_39175124/article/details/79465558)

练习题：请用以下数据（可自行生成尝试，或用其他已有数据集）

1、首先尝试调用sklearn的线性回归函数进行训练；

2、用最小二乘法的矩阵求解法训练数据；

3、用梯度下降法训练数据；

4、比较各方法得出的结果是否一致。

生成数据

In [12]:

```
#生成数据
import numpy as np
#生成随机数
np.random.seed(1234)
x = np.random.rand(500,3)
#构建映射关系，模拟真实的数据待预测值,映射关系为y = 4.2*x0 + 5.7*x1 + 10.8*x2，可自行设置值进行
y = x.dot(np.array([4.2,5.7,10.8]))  #点乘
```

1、先尝试调用sklearn的线性回归模型训练数据

In [59]:

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
%matplotlib inline

# 调用模型
lr = LinearRegression(fit_intercept=True)  #截距项为 True
# 训练模型
lr.fit(x,y)
print("估计的参数值为：%s" %(lr.coef_))
# 计算R平方
print('R^2:%s' %(lr.score(x,y)))
# 任意设定变量，预测目标值
x_test = np.array([2,4,5]).reshape(1,-1)
y_hat = lr.predict(x_test)
print("预测值为：%s" %(y_hat))
```

估计的参数值为：[4.2 5.7 10.8]
R^2:1.0
预测值为：[85.2]

2、最小二乘法的矩阵求解

In [18]:

```
class LR_LS():
    def __init__(self):
        self.w = None
    def fit(self, X, y):
        # 最小二乘法矩阵求解
        #============================== show me your code ======================
        temp = (np.linalg.inv((X.T).dot(X))).dot(X.T)
        self.w = temp.dot(y)
        #============================== show me your code ======================
    def predict(self, X):
        # 用已经拟合的参数值预测新自变量
        #============================== show me your code ======================
        y_pred = X.dot(self.w)
        #============================== show me your code ======================
        return y_pred

if __name__ == "__main__":
    lr_ls = LR_LS()
    lr_ls.fit(x,y)
    print("估计的参数值：%s" %(lr_ls.w))
    x_test = np.array([2,4,5]).reshape(1,-1)
    print("预测值为：%s" %(lr_ls.predict(x_test)))
```

估计的参数值：[4.2 5.7 10.8]
预测值为：[85.2]

3、梯度下降法

In [58]:

```
class LR_GD():
    def __init__(self):
        self.w = None
    def fit(self,X,y,alpha=0.02,loss = 1e-10): # 设定步长为0.002,判断是否收敛的条件为1e-1
        y = y.reshape(-1,1) #重塑y值的维度以便矩阵运算
        [m,d] = np.shape(X) #自变量的维度
        self.w = np.zeros((d)) #将参数的初始值定为0
        tol = 1e5
        #============================== show me your code ======================
        while tol > loss:  #退出条件：两次 loss 差值
            f = np.dot(X, self.w).reshape(-1, 1)  #必须重塑为 1 列，与 y 运算
            error = y - f
            self.w += alpha*np.mean((X*error), axis=0) #axis=0表示：输出矩阵是1行，求每-
            tol = np.sqrt((error ** 2).mean()) #rmse均方误差

        #============================== show me your code ======================
    def predict(self, X):
        # 用已经拟合的参数值预测新自变量
        y_pred = X.dot(self.w)
        return y_pred

if __name__ == "__main__":
    lr_gd = LR_GD()
    lr_gd.fit(x,y)
    print("估计的参数值为：%s" %(lr_gd.w))
    x_test = np.array([2,4,5]).reshape(1,-1)
    print("预测值为：%s" %(lr_gd.predict(x_test)))
```

估计的参数值为：[4.2 5.7 10.8]
预测值为：[85.2]

参考

吴恩达 CS229课程

周志华 《机器学习》

李航 《统计学习方法》

https://hangzhou.anjuke.com/ (https://hangzhou.anjuke.com/)

https://www.jianshu.com/p/e0eb4f4ccf3e (https://www.jianshu.com/p/e0eb4f4ccf3e)

https://blog.csdn.net/qq_28448117/article/details/79199835
(https://blog.csdn.net/qq_28448117/article/details/79199835)

https://blog.csdn.net/weixin_39175124/article/details/79465558
(https://blog.csdn.net/weixin_39175124/article/details/79465558)

In [ ]:

```
```