



今日头条



转发



微博



Qzone



微信

首页 / 技术 / 正文

搜索

求解线性回归应使用哪些方法？

计算机与AI 2020-09-13 13:55:52

作为任何机器学习工具箱中的基础算法集，可以使用多种方法来解决线性回归问题。在这里，我们讨论。结合代码示例，四种方法，并演示应如何使用它们。

线性回归是一种受监督的机器学习算法。它基于给定的因变量（x）预测自变量（y）之间的线性关系，以使自变量（y）的 **成本最低**。

► 解决线性回归模型的不同方法

为了提高效率，可以将许多方法应用于线性回归模型。但是，我们将在这里讨论其中最常见的问题。

1. 梯度下降
2. 最小二乘法/正态方程法
3. Adam
4. 奇异值分解（SVD）

好吧，让我们开始吧...

► 梯度下降

对于初学者来说，解决线性回归问题的最常见，最简单的方法之一就是梯度下降。

梯度下降的工作原理

现在，让我们假设我们以散点图的形式绘制了数据，并且当我们对它应用成本函数时，我们的模型将做出预测。现在，此预测可能非常好，或者可能与我们的理想预测相去甚远（这意味着



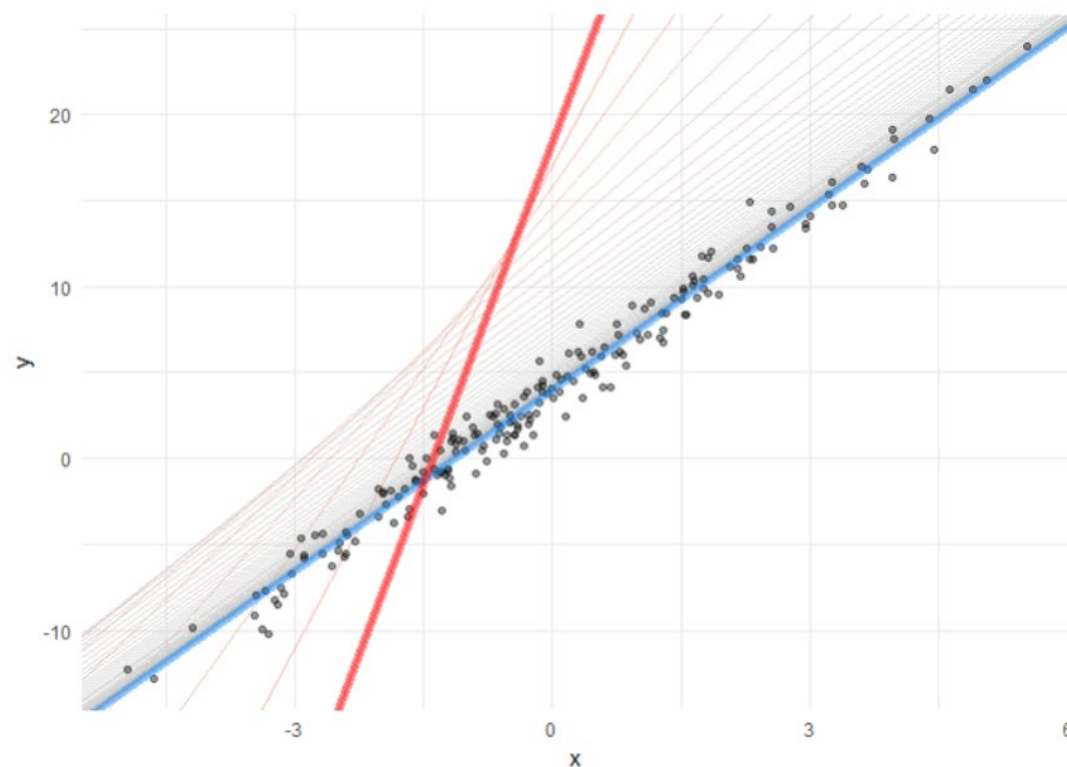
计算机与AI

关注

- 求解线性回归应使用哪些方法？
- 如何评估机器学习模型的性能

其成本将会很高)。因此，为了最小化该成本（误差），我们对其应用了梯度下降。

现在，梯度下降将使我们的假设逐渐收敛到**成本** 最低的全局最小值。为此，我们必须手动设置**alpha**的值，并且假设的斜率相对于我们alpha的值而变化。如果alpha值较大，则将采取较大步骤。否则，在小阿尔法的情况下，我们的假设将缓慢收敛并通过小步伐收敛。



梯度下降公式为

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta).$$

在Python中实现梯度下降

```
import numpy as np
from matplotlib import pyplot
```

```
#creating our dataX = np.random.rand(10,1)
y = np.random.rand(10,1)
m = len(y)
theta = np.ones(1)
#applying gradient descenta = 0.0005
cost_list = []for i in range(len(y)):
    theta = theta - a*(1/m)*np.transpose(X)@(X@theta - y)
    cost_val = (1/m)*np.transpose(X)@(X@theta - y)
    cost_list.append(cost_val)
#Predicting our Hypothesisb = thetayhat = X.dot(b)#Plotting our
resultspyplot.scatter(X, y, color='red')
pyplot.plot(X, yhat, color='blue')
pyplot.show()
```

首先，我们创建了数据集，然后遍历了所有训练示例，以最大程度地降低假设成本。

优点：

梯度下降的重要优点是

- 与SVD或ADAM相比，计算成本更低
- 运行时间为 $O(kn^2)$
- 与更多功能一起使用效果很好

缺点：

梯度下降的重要缺点是

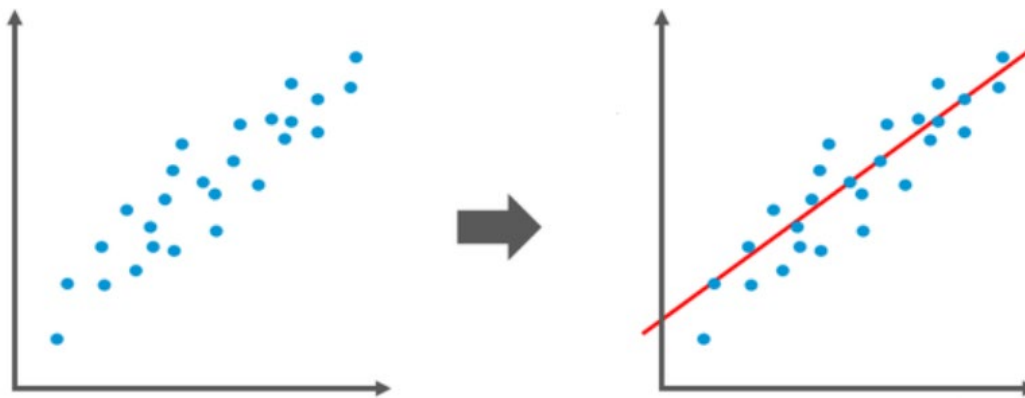
- 需要选择一些学习率 α
- 需要多次迭代才能收敛
- 可以卡在本地最小值
- 如果学习率 α 不合适，则可能无法收敛。

► 最小二乘法

最小二乘法也称为 **正态方程**，也是轻松求解线性回归模型的最常用方法之一。但是，这需要具有线性代数的一些基本知识。

最小二乘法如何工作

在普通的LSM中，我们直接求解系数值。简而言之，我们一步就能达到光学的最低点，或者我们只能说一步就可以以最低的成本使我们的假设适合我们的数据。



LSM的公式是

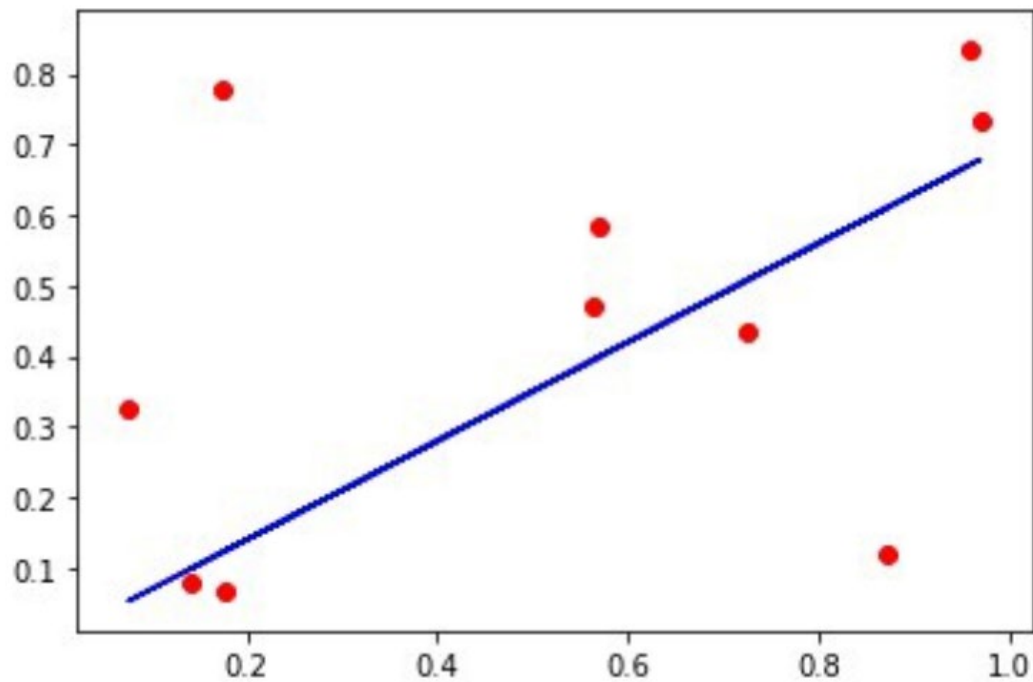
$$\theta = ((X^t X)^{-1}). (X^t y)$$

在Python中实现LSM

```
import numpy as np
from matplotlib import pyplot
#creating our dataX = np.random.rand(10,1)
y = np.random.rand(10,1)
#Computing coefficientb =
np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)#Predicting our Hypothesisyhat
= X.dot(b)#Plotting our resultspyplot.scatter(X, y, color='red')
pyplot.plot(X, yhat, color='blue')
```

求解线性回归应使用哪些方法？

```
pyplot.show()
```



在这里，我们首先创建了数据集，然后使用

$$b = \text{np.linalg.inv}(X^T \cdot X) \cdot \text{dot}(X^T, y)$$

优点：

LSM的重要优点是：

- 没有学习率
- 没有迭代
- 不需要功能缩放
- 当“功能数量”较少时，效果很好。

缺点：

重要的缺点是：

- 当数据集很大时，计算量很大。
- 功能数量多时速度慢
- 运行时间为O（n³）
- 有时，您的X转置X是不可逆的，即，没有逆的奇异矩阵。您可以使用np.linalg.pinv代替 np.linalg.inv 来解决此问题。

► Adam方法

ADAM代表自适应矩估计，是一种在深度学习中广泛使用的优化算法。

这是一种迭代算法，适用于嘈杂的数据。

它是RMSProp和小批量梯度下降算法的组合。

除了存储像Adadelata和RMSprop这样的过去平方梯度的指数衰减平均值之外，Adam还保留了过去梯度的指数衰减平均值，类似于动量。

我们分别计算过去和过去平方梯度的衰减平均值，如下所示：

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned}$$

当 **mt**和 **vt**初始化为0的向量时，Adam的作者观察到它们偏向零，特别是在初始时间步长中，尤其是在衰减率较小时（即β1β1和β2β2接近1）。

他们通过计算经过偏差校正的第一和第二矩估算值来抵消这些偏差：

求解线性回归应使用哪些方法？

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

然后，他们使用以下命令更新参数：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

```
#Creating the Dummy Data set and importing libraries
import mathimport seaborn as sns
import numpy as np
from scipy import stats
from matplotlib import pyplot
x = np.random.normal(0,1,size=(100,1))y = np.random.random(size=
(100,1))
```

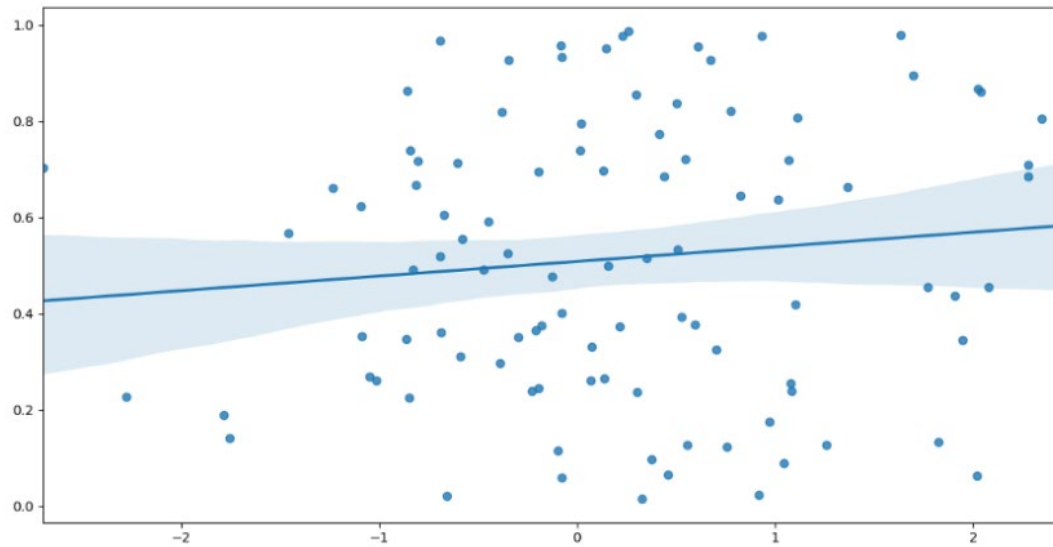
现在，让我们找到线性回归的实际图形以及数据集的斜率和截距值。

```
print("Intercept is ",stats.mstats.linregress(x,y).intercept)
print("Slope is ", stats.mstats.linregress(x,y).slope)
```

```
Intercept is  0.5090649899143213
Slope is  0.030360374593297913
```

现在，让我们使用Seaborn **regplot** 函数查看线性回归线。

```
pyplot.figure(figsize=(15,8))
sns.regplot(x,y) pyplot.show()
```



```

h = lambda theta_0, theta_1, x: theta_0 + np.dot(x, theta_1) #equation
of straight lines
# the cost function (for the whole batch. for comparison later)
def J(x, y, theta_0, theta_1):    m = len(x)    returnValue = 0
    for i in range(m):
        returnValue += (h(theta_0, theta_1, x[i]) - y[i])**2
    returnValue = returnValue/(2*m)
    return returnValue
# finding the gradient per each training example
def grad_J(x, y, theta_0, theta_1):    returnValue = np.array([0., 0.])
    returnValue[0] += (h(theta_0, theta_1, x) - y)
    returnValue[1] += (h(theta_0, theta_1, x) - y)*x
    return returnValue
class AdamOptimizer:    def __init__(self, weights, alpha=0.001,
beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.alpha = alpha
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon

```



```
self.m = 0
self.v = 0
self.t = 0
self.theta = weights
    def backward_pass(self, gradient):
self.t = self.t + 1
self.m = self.beta1*self.m + (1 - self.beta1)*gradient
self.v = self.beta2*self.v + (1 - self.beta2)*(gradient**2)
m_hat = self.m/(1 - self.beta1**self.t)
v_hat = self.v/(1 - self.beta2**self.t)
self.theta = self.theta - self.alpha*(m_hat/(np.sqrt(v_hat) -
self.epsilon))
    return self.theta
```

在这里，我们使用面向对象的方法和一些辅助函数来实现上述伪代码中提到的所有方程式。

现在让我们为模型设置超参数。

```
epochs = 1500
print_interval = 100
m = len(x)
initial_theta = np.array([0., 0.]) # initial value of theta, before
gradient descent
initial_cost = J(x, y, initial_theta[0], initial_theta[1])
theta = initial_thetaadam_optimizer = AdamOptimizer(theta, alpha=0.001)
adam_history = [] # to plot out path of descent
adam_history.append(dict({'theta': theta, 'cost': initial_cost})#to
check theta and cost function
```

最后是训练过程。

```
for j in range(epochs):
    for i in range(m):
        gradients = grad_J(x[i], y[i], theta[0], theta[1])
```

```

        theta = adam_optimizer.backward_pass(gradients)
    if ((j+1)%print_interval == 0 or j==0):
        cost = J(x, y, theta[0], theta[1])
        print ('After {} epochs, Cost = {}, theta = {}'.format(j+1,
cost, theta))
        adam_history.append(dict({'theta': theta, 'cost': cost}))
        print ('\nFinal theta = {}'.format(theta))

```

```

After 1 epochs, Cost = [0.131575], theta = [0.0828198  0.02417232]
After 100 epochs, Cost = [0.04011889], theta = [0.50742288  0.02921758]
After 200 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 300 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 400 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 500 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 600 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 700 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 800 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 900 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 1000 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 1100 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 1200 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 1300 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 1400 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]
After 1500 epochs, Cost = [0.04011889], theta = [0.50742285  0.02921755]

Final theta = [0.50742285  0.02921755]

```

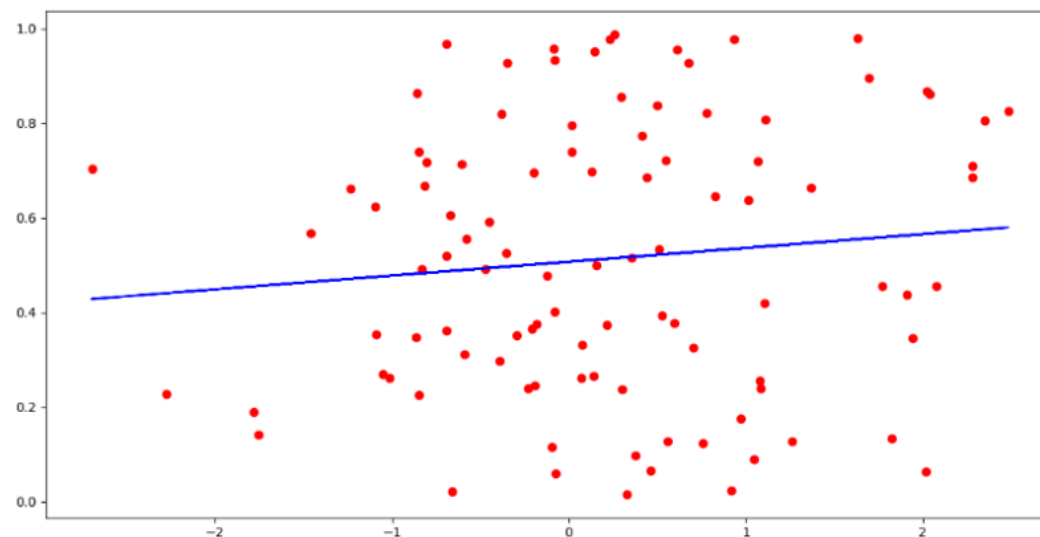
现在，如果将 **Final theta** 值与先前使用`scipy.stats.mstat.linregress`计算的斜率和截距值进行比较，则它们几乎相等，并且可以通过调整超参数来达到100%相等。

最后，让我们绘制它。

```

b = theta
yhat = b [0] + x.dot (b [1])
pyplot.figure (figsize = (15,8) )
pyplot.scatter (x, y, color ='red')
pyplot.plot (x , yhat, color ='blue')
pyplot.show ()

```



我们可以看到我们的绘图类似于使用`sns.regplot`获得的 **绘图**。

优点：

- 直接实施。
- 计算效率高。
- 很少的内存需求。
- 梯度的对角线重标不变。
- 非常适合于数据和/或参数较大的问题。
- 适用于非固定目标。
- 适用于非常嘈杂/或稀疏梯度的问题。
- 超参数具有直观的解释，通常需要很少的调整。

缺点：

- Adam和RMSProp对某些学习率值（有时还包括批处理大小等其他超参数）高度敏感，如果学习率太高，它们可能会灾难性地无法收敛。

► 奇异值分解

由于SVD是线性回归中最著名且使用最广泛的降维方法之一，因此缩短了奇异值分解。

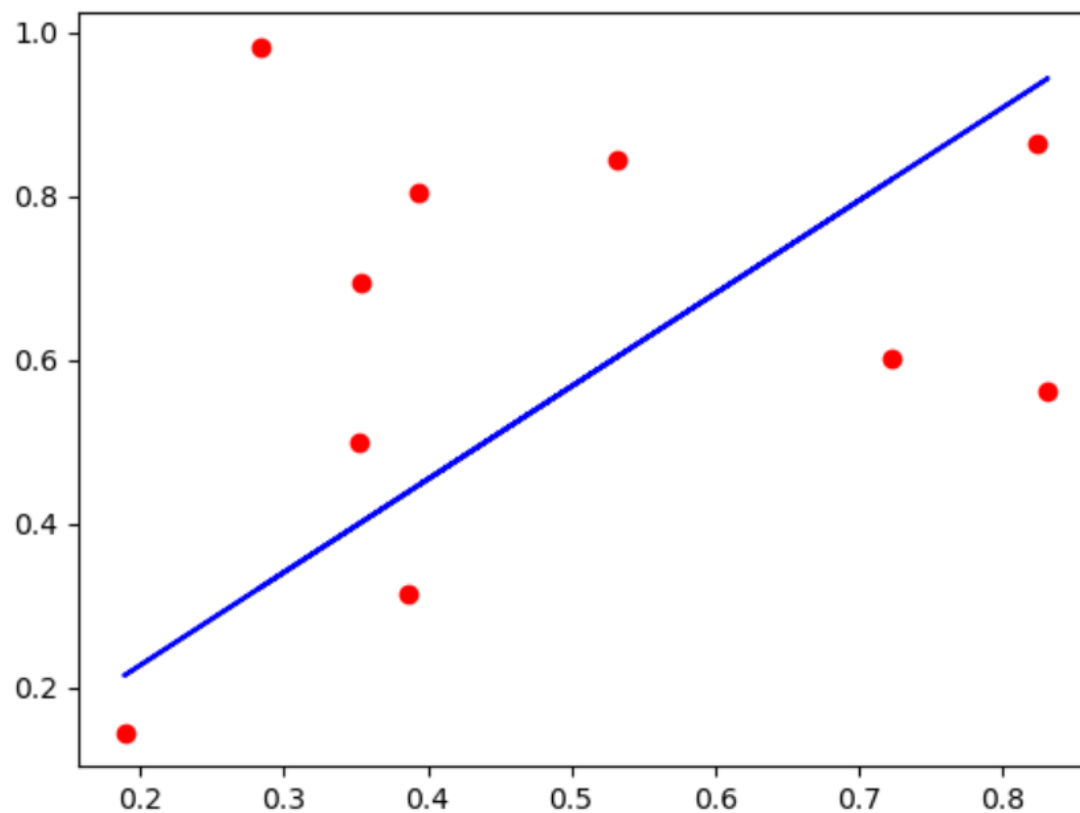
SVD（作为其他用途）被用作预处理步骤，以减少我们的学习算法的维数。SVD将矩阵分解为其他三个矩阵（U，S，V）的乘积。

$$\underset{m \times n}{X} = \underset{m \times m}{U} \underset{m \times n}{\Sigma} \underset{n \times n}{V}^T$$

矩阵分解后，可以通过计算输入矩阵**X**的伪逆 并将其乘以输出向量 **y**来找到假设的系数 。在那之后，我们使我们的假设适合我们的数据，这使我们的成本最低。

```
import numpy as np
from matplotlib import pyplot
#Creating our dataX = np.random.rand(10,1)
y = np.random.rand(10,1)
#Computing coefficientb = np.linalg.pinv(X).dot(y)#Predicting our
Hypothesisyhat = X.dot(b)#Plotting our resultspyplot.scatter(X, y,
color='red')
pyplot.plot(X, yhat, color='blue')
pyplot.show()
```

求解线性回归应使用哪些方法？



尽管它不能很好地融合在一起，但是仍然相当不错。

在这里，首先，我们创建了数据集，然后使用 $b = \text{np.linalg.pinv}(X) \cdot y$ （这是SVD的等式）来最小化假设的成本。

优点：

- 使用更高维度的数据时效果更好
- 适用于高斯型分布式数据
- 对于小型数据集，真正稳定且高效
- 在求解线性回归线性方程时，它是更稳定且首选的方法。

缺点：

运行时间为O (n³)

- 多重危险因素
- 对异常值非常敏感
- 数据集很大时可能会变得不稳定

▶ 学习成果

到目前为止，我们已经学习并实现了梯度下降，LSM，ADAM和SVD。现在，我们对所有这些算法都有很好的了解，并且我们也知道有什么优缺点。


我们注意到的一件事是，ADAM优化算法是最准确的，根据实际的ADAM研究论文，ADAM几乎胜过所有其他优化算法。

☐ 收藏 ☐ 举报

3 条评论




评论

 GEO世界 11小时前

值得赞💎💎💎💎 使用 Python 做数学就是那么简单，人人都应该使用 Python 学数学和物理，甚至应从初中生就开始 ...

回复


0 ☐ ☐

 学术上的那些事 5小时前

学习资料

回复

0 ☐ ☐

 lolybob 23小时前

转发了

回复

0 ☐ ☐

相关推荐



SQLZoo：练习SQL的最佳方法

闻数起舞 · 7评论 · 相关

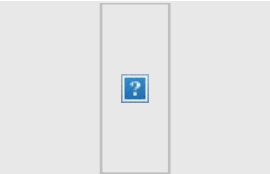
不愿



模板方法模式，看JDK和Spring是如何优雅复用代码的

JavaKeeper · 7评论 · 相关

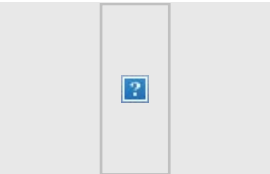
不愿



多模态数据的主题建模：自回归方法

慕测科技 · 5评论 · 相关

不愿



EfficientNet模型的完整细节

AI公园 · 5评论 · 相关

不愿



开源软件分享-漂亮的WPF UI界面框架

IT点滴 · 48评论 · 相关

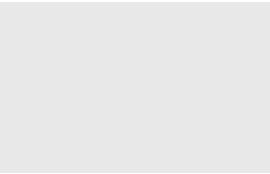
不愿



亿万级海量数据去重软方法，spark/flink/mr等通用

追逐仰望星空 · 22评论 · 相关

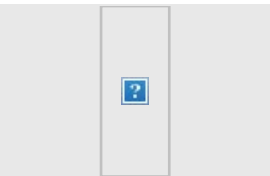
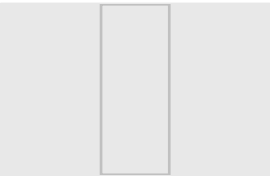
不愿



MySQL--更高效的mysql模糊查询的方法

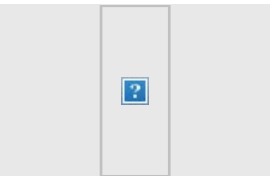
编程仔日常 · 11评论 · 相关

不愿



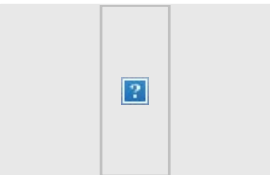
异步社区 · 5评论 · [相关](#)

不愿



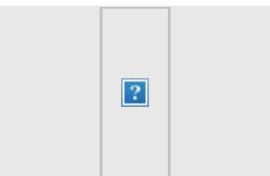
程序员高级码农II · 16评论 · [相关](#)

不愿



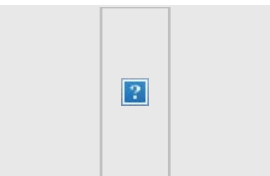
深度学习与NLP · 77评论 · [相关](#)

不愿



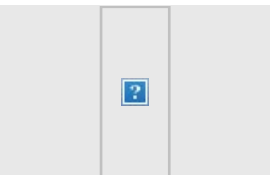
机器学习与数据分析 · 4评论 · [相关](#)

不愿



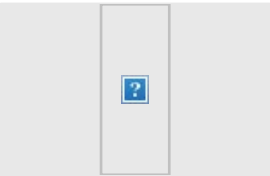
一只野生程序猿 · 12评论 · [相关](#)

不愿



o月牙数据库架构师o · 36评论 · [相关](#)

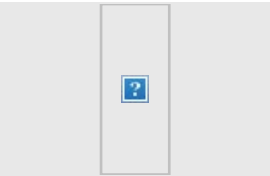
不愿



CPU中的程序是怎么运行起来的（预告篇）

 良知犹存 · 1评论 · [相关](#)

不愿



在Deno中构建一个命令行天气预报程序

 杭州程序员小张 · 6评论 · [相关](#)

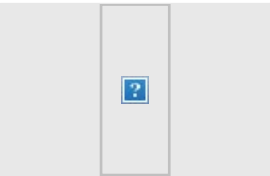
不愿




Vue实战083：几种常用的外部JS文件定义和引用方法详解

 编程手札 · 2评论 · [相关](#)

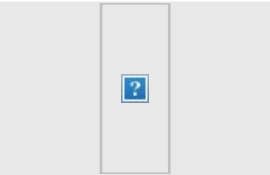
不愿



MicroPython 玩转硬件系列6：获取天气情况

 TopSemic · 3评论 · [相关](#)

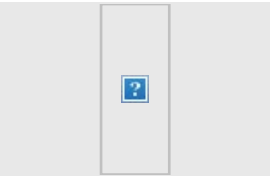
不愿



动作识别与关系推理

 人工智能前沿学生论坛 · 9评论 · [相关](#)

不愿



3分钟短文：Laravel请求对象方法极多，可不是花拳绣腿

 程序员小助手 · 1评论 · [相关](#)

不愿



MATLAB数据分析，基于神经网络河南省降水量预测

 大话数据分析 · 6评论 · [相关](#)

不愿