【猫猫的机器学习-1】从初中数学开始的Linear Regression

Makiror Ouyang

文章封面的抹布来自一个令我印象深刻的数学课发生的事，正好是讲函数的，写文章时莫名联想到了。玩梗罢了，它并没有真扔到我脸上（只是桌上）一笑而过就好。

引言

等中考结束后我也差不多期末考了，这段时间算是思绪很乱，学习效率很低下，在迫不得已的焦虑下整了本*Introduction to machine learning*《机器学习导论》，对这玩意有点兴趣和冲动后就去看了很多有关的东西，我觉得我还是更适合从实际出发吧，所以就决定开这个系列。

正如标题所写，这个新系列，我希望做的是在保证质量的前提下尽可能让内容更好理解。我还是一如既往的那个不擅长理科的Makiror，自己也有些恶补缺失基础和一步步学习的经验，所以你可以比较不用担心我的Blog会出现令人费解的高深内容，或是没必要的，故弄玄虚的花里胡哨。

本文更偏向理论，没有假设读者门槛，如果有什么看不懂的地方顺道补上就好了，第一篇不会难的。文章内有少量示例，使用Haskell编程语言编写。

要注意的是，本文很多知识点如果展开可以专门写一篇文章了，所以这次更倾向于梗概，之后有时间我会为部分比较重要的东西专门写博客。

基础概念

因为是第一篇，考虑到不知道机器学习的读者，在正文开始之前我们先来了解一些机器学习有关的基础概念。

什么是机器学习?

机器学习（Machine Learning）是一门关注如何让计算机通过数据学习和自动改进的领域，它已经在各个行业和领域产生了巨大的影响。随着计算能力的提升和大数据的兴起，机器学习技术变得越来越重要，被广泛应用于自动驾驶、语音识别、推荐系统、金融预测等众多领域。

在传统的计算机程序中，我们需要明确地表示规则和逻辑来完成特定任务，例如图像识别、语音识别或自然语言处理。但是在面对复杂的、高度变化的现实世界问题时，这样使得我们的任务变得非常困难甚至不可行，而机器学习就是让计算机从大量的数据中进行学习，并通过统计学方法和一些算法来自动发现数据中的模式和规律。它的目标是构建能够从数据中进行学习、泛化并做出预测或决策的模型，这些模型可以根据已有的数据进行训练和优化，然后利用学到的知识来对新的、未见过的数据进行预测和决策。

学习算法

学习算法是机器学习的核心部分，根据输入数据和所需的任务，自动调整模型的参数或规则，使得模型能够从数据中进行学习和预测。

监督学习是机器学习中的一种算法类型，本篇的『线性回归』就是一种监督学习算法。它是一种通过使用带有标签的训练数据来构建预测模型的算法，模型从已知输入和相应的输出之间的关系中学习，然后根据这个关系对新的输入进行预测。与之对应的无监督模型的数据是没有标签的，算法从数据中发现类别并学习。

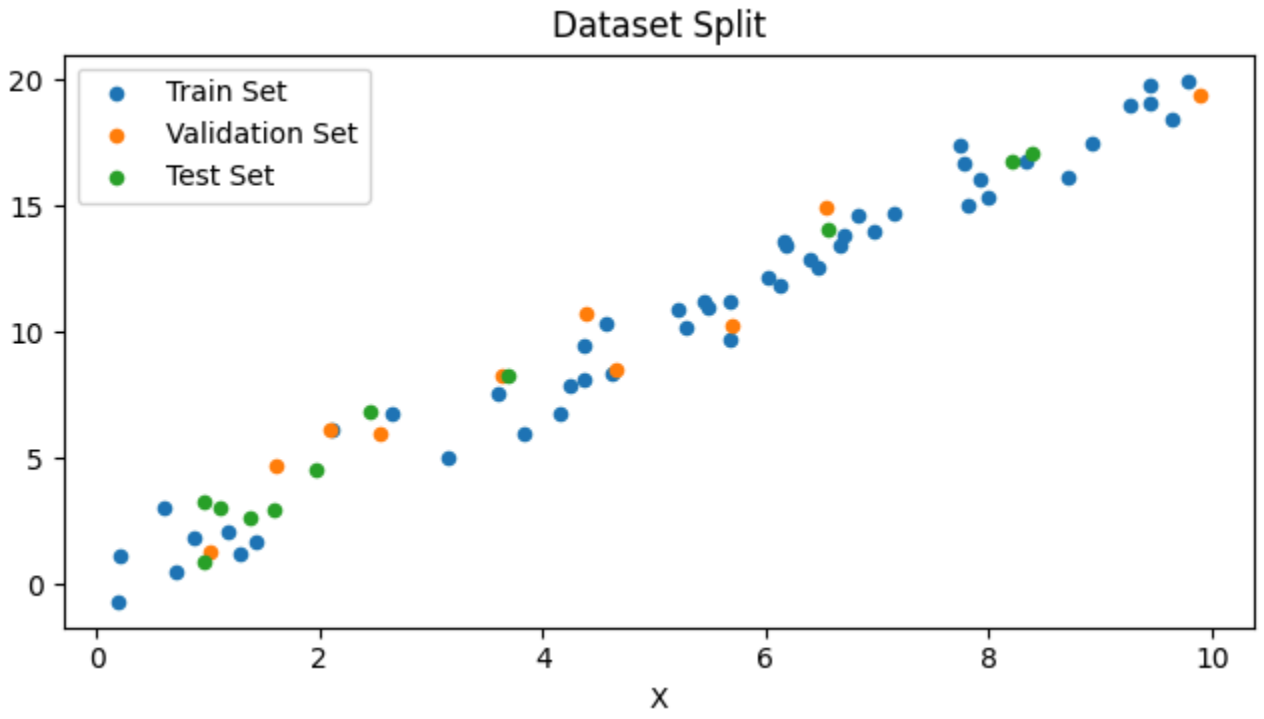
训练集、测试集和验证集

前面提到的“数据”什么的，虽然都这么说，但具体来说在机器学习中的数据也是有划分并区别的，我们有必要先笼统地了解一下。

训练集（Train Set）是用于训练机器学习模型的数据集，在监督学习中，训练集的内容包括样本和对应的输出（也就是我们前面所说的标签），而在无监督模型中，训练集没有对应的输出，仅包含样本。

评估训练后模型的泛化能力（即对未知对应值的数据的预测或者分类能力）、性能等指标的，与训练集相互独立的数据集叫做测试集（Test Set），测试集的样本不包括对应的输出，我们使用测试集来获得对模型性能的评估。

模型训练好后，我们**可以**（这意味着它不是必需的）使用验证集（Validation set）来评估模型的表现，这个概念可能和测试集容易搞混，测试集用于最后评估模型，而验证集用于学习过程中的 *超参数* 调整和选择，验证集的数据是从训练集中划分出来的。



这张散点图是训练集、测试集、验证集的分布示例，其中样本的数量是 训练集 > 测试集 > 验证集。

> 没找到顺手的画图工具，图是用Python的matplotlib写的。

在机器学习算法中，模型的受到多个参数的影响，而我们需要一种可以给我们自己人为控制的参数，这种参数就被叫

做超参数，在训练之前设置的（而不是训练产生的），它存在是为了我们在训练过程中能做些自定义的调整。超参数可以调整模型的复杂度、控制模型的训练过程以及影响模型的泛化能力。

简单线性回归

我们简单了解了一下机器学习是什么，接下来正文开始。

回归分析简介

回归分析是一种统计学方法，用于研究变量之间的关系，并建立一个数学模型来描述和预测因变量与自变量之间的关联，进而用于预测、评估关联性等。这个拟合的数学模型是函数，根据数据特征会再细分成不同类型的函数（例如线性函数、非线性函数）。在回归分析中，我们使用给定的数据来估计回归模型的参数，拟合出回归模型以达到我们的目的。

本文的线性回归是回归分析的一种，说白了就是回归系数要求线性，假设自变量和因变量存在线性关系，我们可以通过一些算法来拟合出这个线性回归模型。

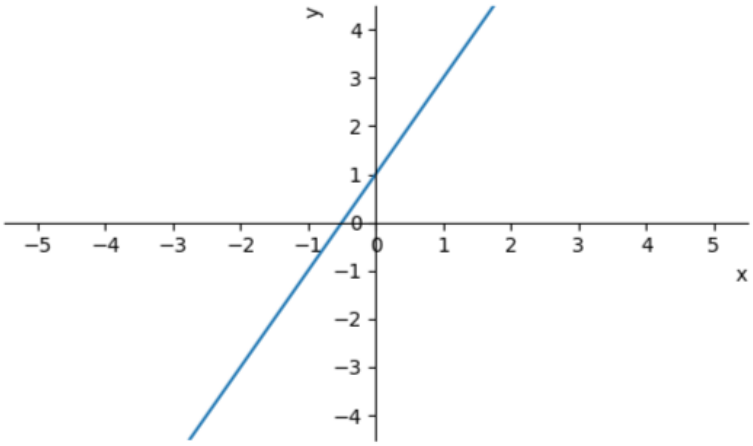
基本模型与表达

所谓简单线性回归（Simple linear regression）就是一元线性回归，只有一个自变量的情况叫做简单回归，我们初中时都学过最简单的一次函数，形如方程：

$$y = kx + b(k \neq 0)$$

其中 x 是自变量，y 是因变量。k 是斜率（slope），决定了自变量 x 对因变量 y 的影响程度，一个常数项。b是截距（intercept），一个常数项。考虑一个简单的一次函数表达式和它的图像：

$$y = 2x + 1$$



Graph of y = 2x + 1

从图像中我们可以看到一次函数的图像呈直线，也就是自变量 x 和因变量 y 呈线性关系。其中当自变量 x 为0时，因变量 y 的值等于截距 b。一元线性回归模型也是如此，自变量和因变量存在线性关系，所以只要有这样的一个函数（即斜率和截距已知），我们就可以通过自变量预测因变量。在线性回归中，截距 b 的意义是基准值，它表示在自变量为0时因变量的基准值，不过它的意义在实际背景下才能做对应的解释，有些时候

自变量为0意味着没有意义（例如用于表示物理量的时候）。准确来说，如果使用 k 代表斜率，b 代表截距，一元线性回归模型应该表示为

$$y = kx + b + \epsilon$$

相比我们熟悉的，它多了一个epsilon，表示无法完全解释的随机误差。

一元线性回归是线性回归中最简单的形式，最直接的前提是自变量和因变量存在线性关系，且每个观测值不会受到其他观测值影响，在后面我们拟合这条直线时还有一些前提会细讲。总而言之，我们只需要拟合出这条直线即可进行预测和推断，而我们接下来将探讨如何找到最佳拟合的斜率和截距。

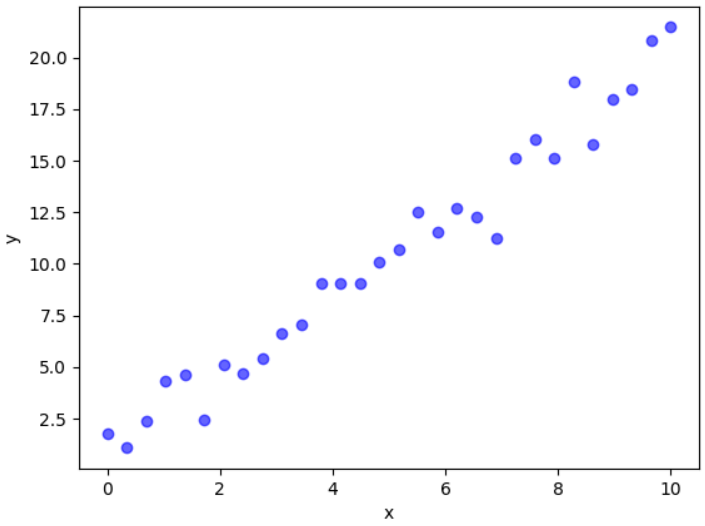
最小二乘法

我们为算法提供一堆二元组作为训练样本（训练集是它们全部的集合的叫法），一个自变量 x 对应一个因变量 y，所以可以这样表示它们。

$$\langle x_0, y_0 \rangle, \langle x_1, y_1 \rangle ..., \langle x_n, y_n \rangle$$

但是样本长什么样都有可能，尤其是在现实应用中，在分析 n 组样本时它们很可能只是大致呈线性分布，而无法找到一条直线能经过所有点，也就是这个方程 $y = kx + b$ 无确定解。

例如在这张例图中，30个样本大致呈线性分布，但很显然它们不是一条直线，没有一个确定的规律可以使我们根据它，确定一个自变量对应的值。但是我们可以尽可能求出一个近似解。



想象一条我们猜测的直线 l 在这个例图上，它与这些点整体的分布方向大致一致（不一定经过所有点），然后再从每个点竖直方向向 l 做线段，这个线段的长度也就是猜测的因变量和实际的因变量的误差，我们要考虑的只是如何猜测这条直线的问题。

我们使用一个“帽子”来表示估计值，这样做是区分估计值和实际值，例如在直线 l 的函数图象中 x 对应的 y 是估计值，所以，我们可以设直线 l 的表达式为 $\hat{y} = kx + b$（还在读初中的读者应该很有亲切感吧，有就对了）。我们可以直接想到误差就是 $(y_i - \hat{y}_i)$，这个误差也叫残差（residual），然后我们求它的平方 $(y_i - \hat{y}_i)^2$，这么做最直接的原因是突出较大误差对整体误差的影响，同时避免正负误差的抵消，就都变成正数了。我们来整理一下：

我们用希腊字母 ε（epsilon）来表示残差

$$\varepsilon_i = y_i - \hat{y}_i$$

对于拟合线 l 的表达式 $\hat{y}=kx+b$，我们可以将 $kx+b$ 代入，并以此展开残差平方和公式。

$$\varepsilon_i^2=y_i-(kx_i+b)$$

$$S=\sum_{i=1}^{n}\varepsilon_i^2\quad=\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\sum_{i=1}^{n}(y_i-(kx_i+b))^2$$

我们要做的就是使 S 最小，我们先看求解斜率 k 的最优值，我们将残差平方和 S 对斜率 k 求导，并使导数等于0，即 $\frac{\partial S}{\partial k}=0$ ，则残差平方和取极小值。这边提一下，在写这一段的时候我有去看一下《人教A版高中数学(选修三册)》对一元线性回归的解法，在教材中不涉及求导，这里稍微有点点超纲了，不过，求导的方法更为通用，后面我们可以直接推广到多元线性回归，我们在简单线性回归的阶段就考虑好后面也更方便，是吧（心虚）

考虑到没有这方面知识的读者，这里简单解释一下求导是什么，导数（derivative）描述了函数在某一点处的变化率或斜率，想象出一个函数图像，求导就是要我们找出在函数某个点上的斜率。

现在你已经有对导数的基本概念了，我们先求解关于斜率 k 和截距 b 的导数。

$$\frac{\partial S}{\partial k}=\frac{\partial}{\partial k}\sum_{i=1}^{n}(y_i-(kx_i+b))^2$$

$$\frac{\partial S}{\partial b}=\frac{\partial}{\partial b}\sum_{i=1}^{n}(y_i-(kx_i+b))^2$$

在这里，我们将 S 视为一个复合函数，表示为 $f(g(k,b))$，内部函数的解析式为 $g(k,b)=y_i-(kx_i+b)$ ，然后我们通过链式法则分别对内部函数和外部函数进行求导，这样我们就能将导数计算拆分成多个简单的函数导数的乘积。我们先对内部函数 g 进行求导。对于 $y_i-(kx_i+b)$ 的 k 求导，其中 y_i 和 b 不含 k项，所以它们的导数为0，而我们只需要考虑 kx_i 项，所以我们对 k 求导的结果是 $-x_i$。我们再对这个内部函数的 b 进行求导，同理，因为截距 b 项是独立的，所以将其他项视为常数项，则对 b 的求导结果为-1。求导如下：

$$\frac{\partial}{\partial k}(y_i-(kx_i+b))=-x_i$$

$$\frac{\partial}{\partial b}(y_i-(kx_i+b))=-1$$

补充，∂ 是偏导数符号，表示函数在某个变量上的变化率，在我们的残差平方和函数中，y_i 是观测值，$\hat{y}_i$ 是预测的因变量值，斜率 k 和截距 b 在这里才是变量，我们在这里已经完成了对于斜率 k 和截距 b 的偏导数计算。

紧接着我们对外部函数进行求导，若内部函数为 $u=y_i-(kx_i+b)$，则外部函数为 $f(u)=u^2$，根据幂函数的导数公式：

若 $f(g)=x^n$，其中 n 是常数，则导数 $\frac{df}{dx}=nx^{n-1}$，根据这个公式，我们将外部函数带进去，即可进行求导。

$$\frac{df}{du}=\sum_{i=1}^{n}2u^{2-1}=\sum_{i=1}^{n}2u=\sum_{i=1}^{n}2(y_i-(kx_i+b))$$

我们先将对 S 关于斜率 k 求偏导的结果代入进去，继续推导并化简。

$$
\begin{aligned}
\frac{\partial S}{\partial k} &= \sum_{i=1}^{n} 2(y_i - (kx_i + b)) \cdot \frac{\partial}{\partial k} \sum_{i=1}^{n} (y_i - (kx_i + b)) \\
&= \sum_{i=1}^{n} 2(y_i - (kx_i + b)) \cdot (-x_i) \\
&= -2 \sum_{i=1}^{n} x_i (y_i - (kx_i + b))
\end{aligned}
$$

同理，将对 S 关于截距 b 求偏导的结果代入进去，推导并化简。

$$
\begin{aligned}
\frac{\partial S}{\partial b} &= \sum_{i=1}^{n} 2(y_i - (kx_i + b)) \cdot \frac{\partial}{\partial b} (y_i - (kx_i + b)) \\
&= \sum_{i=1}^{n} 2(y_i - (kx_i + b)) \cdot (-1) \\
&= -2 \sum_{i=1}^{n} (y_i - (kx_i + b))
\end{aligned}
$$

我们先从截距 b 开始，将对 S 关于截距 b 的偏导数置为0，如下，移项和化简。

$$
\begin{aligned}
-2 \sum_{i=1}^{n} (y_i - (kx_i + b)) &= 0 \\
2 \sum_{i=1}^{n} (kx_i + b - y_i) &= 0 \\
2 \sum_{i=1}^{n} kx_i + 2 \sum_{i=1}^{n} b - 2 \sum_{i=1}^{n} y_i &= 0 \\
\sum_{i=1}^{n} b &= \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} kx_i \\
b &= \frac{\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} kx_i}{n}
\end{aligned}
$$

注意到，x 的平均值为 $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$，$y$ 的平均值为 $\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$，所以

$$
\begin{aligned}
b &= \frac{\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} kx_i}{n} \\
b &= \bar{y} - k\bar{x}
\end{aligned}
$$

我们继续代入推斜率 k，先携带平均值简化后再代入。

$$2\sum_{i=1}^{n}(y_i-(kx_i+b))(-x_i)=0$$

$$2\sum_{i=1}^{n}(kx_i^2+bx_i-x_iy_i)=0$$

$$2\sum_{i=1}^{n}(kx_i^2+(\bar{y}-k\bar{x})x_i-x_iy_i)=0$$

$$\sum_{i=1}^{n}kx_i^2+\sum_{i=1}^{n}\bar{y}x_i-\sum_{i=1}^{n}k\bar{x}x_i-\sum_{i=1}^{n}x_iy_i=0$$

$$k(\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}\bar{x}x_i)=\sum_{i=1}^{n}x_iy_i-\sum_{i=1}^{n}x_i\bar{y}$$

$$k=\frac{\sum_{i=1}^{n}x_iy_i-\sum_{i=1}^{n}x_i\bar{y}}{\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}x_i\bar{x}}$$

$$k=\frac{\sum_{i=1}^{n}x_iy_i-\sum_{i=1}^{n}x_i\frac{\sum_{i=1}^{n}y_i}{n}}{\sum_{i=1}^{n}x_i^2-\sum_{i=1}^{n}x_i\frac{\sum_{i=1}^{n}x_i}{n}}$$

$$k=\frac{n\sum_{i=1}^{n}x_iy_i-\sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i}{n\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2}$$

所以，我们推导出了最佳的斜率 k 和最佳截距 b 的公式。

$$k=\frac{n\sum_{i=1}^{n}x_iy_i-\sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i}{n\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2}$$

$$b=\frac{\sum_{i=1}^{n}y_i-\sum_{i=1}^{n}kx_i}{n}$$

上述的内容是，为了更好读者理解我就自己推了一遍公式（是越推越起劲的这玩意），你没耐心的话只看推导结果即可，别被一大坨数学公式吓着，我们使用Haskell实现这样的算法是很简单的，先让我们定义函数 *unaryLinearRegression* ，它会接受一个类型为Maybe的结果，表示一个可能存在或者不存在的返回值，你可以理解为Rust中的Option，它有两种值：*Just* 和 *Nothing*，假设接收到的是空列表，则返回*Nothing*

```
import Data.List (foldl')

unaryLinearRegression :: [(Double, Double)] -> Maybe (Double, Double)
unaryLinearRegression [] = Nothing
```

首先我们计算样本的数量 n 并赋值变量，然后我们定义一个四元组 $\langle sumX，sumY，sumXY，sumX2\rangle$ 并通过 *fold'*函数将累加函数 *accumulate* 应用于每个样本，这四元组的元素分别是样本 x 坐标和，样本 y 坐标和，每个样本的 xy 乘积和，每个样本x 坐标平方和。 *accumulate* 接受四元组和一个样本并返回新的四元组，通过定义运算 *(sX + x, sY + y, sXY + x * y, sX2 + x * x)* 将其传给下一个样本处理。最后我们根据上面推导出来的公式计算斜率和截距并返回结果即可。另外写一个函数将结果转为只有3为小数的浮点数。

```
formatFloat :: Double -> Double
formatFloat x = fromIntegral (round (x * 1000) :: Integer) / 1000

unaryLinearRegression points =
  let n = fromIntegral $ length points
      (sumX, sumY, sumXY, sumX2) = foldl' accumulate (0, 0, 0, 0) points
      accumulate (sX, sY, sXY, sX2) (x, y) = (sX + x, sY + y, sXY + x * y, sX2 + x * x)
      slope = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX)
      intercept = (sumY - slope * sumX) / n
  in Just (formatFloat slope, formatFloat intercept)
```

然后我们写一个简单的函数，提供斜率、截距和X即可计算预测的值 $\hat{y}$。

```
&amp;nbsp;&amp;nbsp;predict :: (Double, Double) -> Double -> Double
predict (slope, intercept) x = formatFloat intercept + slope * x
```

或许我们可以尝试写点什么测试一下，我提供一些很明显关系为 $y = 2x$ 的样本，让算法来拟合出这个函数解析式并为我们预测另外一个值。

```
module Main (main) where
import LinearRegression

main :: IO ()
main = do
  let dataset = [(1, 2), (2, 4), (3, 6), (4, 8), (5, 10)]
  let regressionResult = unaryLinearRegression dataset
  case regressionResult of
    Just result -> do
      let x = 8
          predictedValue = predict result x
      putStrLn $ "Linear regression result: " ++ show result
      putStrLn $ "Predicted value for x = " ++ show x ++ ": " ++ show predictedValue
    Nothing ->
      putStrLn "Linear regression failed"
```

得到结果：

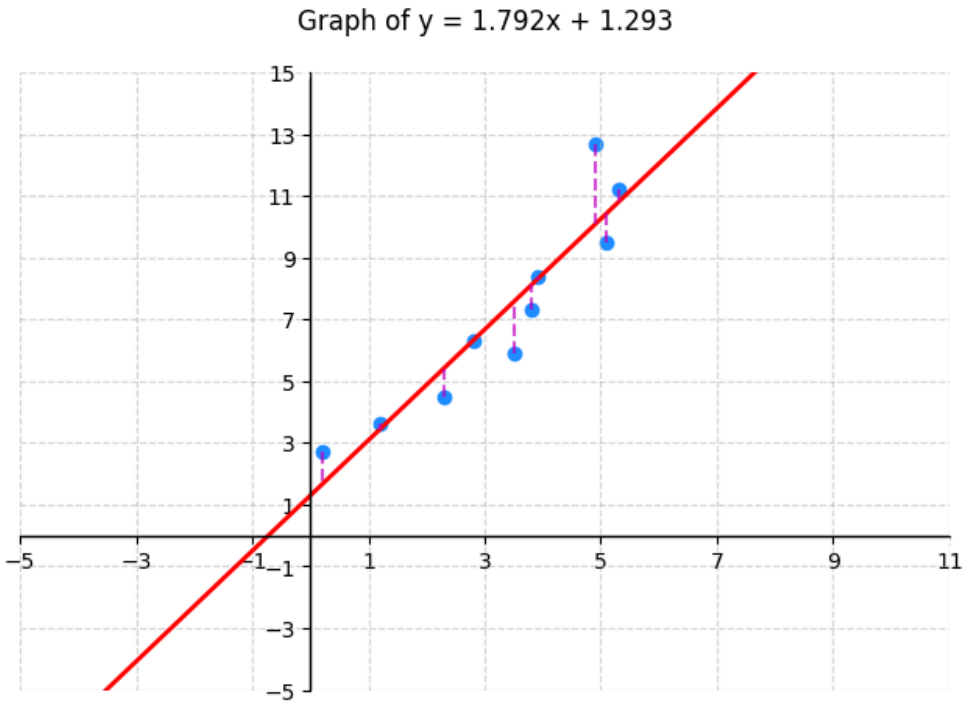
```
Linear regression result: (2.0,0.0)
Predicted value for x = 8.0: 16.0
```

然后我让AI为我生成一组10个，部分带有小数的，大致呈线性关系的样本，如下。

```
[(0.2, 2.7), (1.2, 3.6), (2.3, 4.5), (2.8, 6.3), (3.5, 5.9), (3.8, 7.3), (3.9, 8.4), (5.1, 9.5), (5.3, 11.2), (4.9, 12.7)]
```

同时我将x设为4，作为程序输入，得到的输出如下（保留三位小数）

```
Linear regression result: (1.792,1.293)
Predicted value for x = 4.0: 8.464
```



AI在生成训练集上做的不太好，尽管我强调样本不应该可以被一条直线经过，需要具有一定随机性，它给我跑出来的还是几个可以一条直线经过的点，这个训练集是经过我自己调整的。我使用MatplotLib画出了对应的函数图象和训练集的散点图，观察图像和预测的大致位置，其中紫色虚线则是每个样本的残差。

我们从最简单的初中一次函数了解了一元线性回归，并且简单地推导出了相关公式并实现了这样的算法，现在我们要推广到更复杂的情况——多元线性回归。别急，除此以外我们还有一些拓展章。

多元线性回归

进入多元线性回归内容会更偏理论，因为在实际情况确定的情况下拟合的方法也就那样，但是『多元线性回归』这个词本身涉及很多可能性，而不是单纯一条直线，所以我们没法使用一种实际的、具体的实例来概括所有。而我尽可能用偏理论的表达使内容更通用（例如在举例二元线性回归时，没有特地强调的地方就是可以推广到更多元的）。在本章我们仅需了解多元线性回归的表达和公式推导即可，而某些和多元线性回归有一定关联的东西会放在后面的章节。

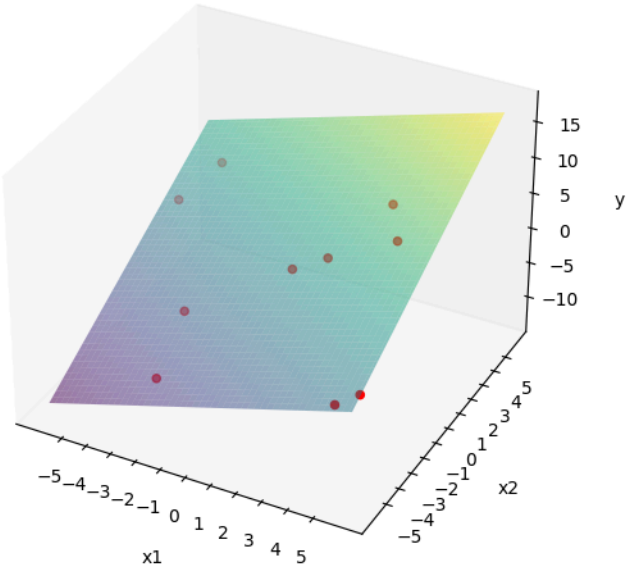
基本模型与表达

不同于一元线性回归，多元线性回归（Multiple Linear Regression）有了多个自变量，之前我们用了初中生最熟悉的一次函数表达的方式（我个人其实更习惯希腊字母），现在为了方便，换成更广泛被使用的 β，对于多元线性回归

模型，表示为

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \varepsilon$$

一样，y 是因变量，x_n 是自变量，β_n是回归系数，ϵ是随机误差，在后面我们的学习中，有时会把 $\beta_0, \beta_1, ..., \beta_n$ 统称为模型参数。既然都是线性回归，多元线性回归的因变量与多个自变量也有线性关系，且每个自变量都有一个对应的回归系数。与一元线性回归相比，多元线性回归的最优解不是一条直线，而是一个多维空间中的超平面，例如这样。



在图例中，x_1 和 x_2 是自变量，y 是因变量，y 是根据两个自变量的线性组合计算得到的，所以这是一个二元线性回归，我们管图里的这个平面叫回归面。同理，这个面也是通过样本来拟合出的最佳平面。图例的二元线性回归模型表示为 $y = x_1 + 1.5x_2 + 2$。

我们可以用矩阵表达一个二元线性回归模型，表达式为：

$$y = X\beta + \varepsilon$$

其中 y 是 $n * 1$ 的响应变量向量，X 是一个 $n * (m + 1)$ 的矩阵，包含了 n 个样本的 m 个自变量和一列常数项（截距），β 是一个 $(m + 1) * 1$ 的回归系数向量，包含每个自变量的系数和常数项的系数，ε 是误差。

还是考虑到没有向量和矩阵基础的读者，简单介绍一下，向量（Vector）就是一组有序排列的数值，在不同的领域有不同的含义，在此做出我们用来单纯表示一组值。矩阵（Matrix）可以看做向量的拓展，其中包含多个向量。

我按照上面的例子随机在图中搞了10个散点，虽然它是按照 $y = x_1 + 1.5x_2 + 2$ 的y值偏差为+-0.5来生成的，但是因为这个平面不是用这10个点通过算法拟合出来的，不够严谨的，此处仅作矩阵表示示例。

```
(-5.3, 2.4, 0.6) (-2.6, -2.1, -4.1) (0.9, -0.5, 2.3) (2.1, -0.1, 4.3)
(-4.4, 4.1, 3.7) (4.5, 0.5, 7.9) (5.3, -5.9, -1.0) (6.0, -5.4, 0.1)
(3.5, 2.2, 8.8) (-2.2, -4.9, -7.6)
```

按照上面的定义，我们可以得出

$$\mathbf{X} = \begin{bmatrix} -5.3 & 2.4 \\ -2.6 & -2.1 \\ -0.9 & -0.5 \\ 2.1 & -0.1 \\ -4.4 & -4.1 \\ 4.5 & 0.5 \\ 5.3 & -5.9 \\ 6.0 & -5.4 \\ 3.5 & 2.2 \\ -2.2 & -4.9 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0.6 \\ -4.1 \\ 2.3 \\ 4.3 \\ 3.7 \\ 7.9 \\ -1.0 \\ 0.1 \\ 8.8 \\ -7.6 \end{bmatrix} \quad \beta = \begin{bmatrix} 2 \\ 1 \\ 1.5 \end{bmatrix}$$

此处无法解释的误差 ε 我们暂且忽略，在那之前我们还有更重要的要考量，这个矩阵表达式为 $y = X\beta$，但是这个多元线性回归是有假设的。碍于计算较麻烦，随机生成的数据和示例是比较不严谨的，所以上面的矩阵表示仅作示例，不会继续当成例子来沿用。

最小二乘法

好，我们现在要将最小二乘法推广到多元线性回归，也就是使得求出的超平面的残差平方和最小。我们开始进行一个简单的推导，基于我们之前的思路，我们将目标函数**改为向量形式**，首先看我们之前的残差（推广到向量形式后就不需要下标了，这是整个的）：

$$\varepsilon = y - \hat{y}$$

根据 $y = X\beta + \varepsilon$ 的变形，我们可以得到 $\varepsilon = y - X\beta$，所以就是 $\varepsilon = y - \hat{y} = y - X\beta$。

现在我们需要得到残差平方和公式，一般情况我们默认把 ε 视为列向量，所以我们需要将向量 ε 先转置为行向量，也就是这样得出残差平方和 S 的等式。在转置后，$\varepsilon^\top$ 是 $1 \times n$ 的行向量，而没转置的 ε 是 $n \times 1$ 的列向量，经过运算我们得到的是一个 1×1 的标量。

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ ... \\ \varepsilon_n \end{bmatrix} \quad \varepsilon^\top = \begin{bmatrix} \varepsilon_1, \varepsilon_2, ..., \varepsilon_n \end{bmatrix}$$

$$S = \varepsilon^\top \varepsilon$$

将 $\varepsilon = y - X\beta$ 代入 S，我们得到如下式子，并对其展开：

$$\begin{aligned} S &= (y - X\beta)^\top (y - X\beta) \\ &= (y^\top - (\beta^\top X^\top))(y - X\beta) \\ &= y^\top y - 2\beta^\top X^\top y + \beta^\top X^\top X\beta \end{aligned}$$

我们要对上公式中的 β 求导，将导数置为0，记为 $\dfrac{dS}{d\beta} = 0$，由于第一项 $y^\top y$ 是常数向量，所以求导为0，我们有

$$\frac{dS}{d\beta} = y^\top y - 2\beta^\top X^\top y + \beta^\top X^\top X\beta$$
$$= 0 - 2\beta^\top X^\top y + \beta^\top X^\top X\beta$$

进行移项和推导，得到

$$\begin{aligned} -2X^\top y + 2X^\top X\beta &= 0 \\ 2X^\top X\beta &= 2X^\top y \\ X^\top X\beta &= X^\top y \\ \beta &= (X^\top X)^{-1}X^\top y \end{aligned}$$

所以，这就是我们得到多元线性回归最佳系数 β 向量的公式。

多元共线性

在多元线性回归，我们的自变量就不止一个了，多元共线性（Multicollinearity）是指自变量之间存在相关的情况，一般而言适度的多元共线性问题不大，但是当自变量之间高度相关时回归模型的构建就会受到影响，导致分析结果不稳定等情况。原本自变量是各自独立的，如果它们存在高度相关性则会无法确定其他变量，算法很难找到自变量和因变量之间的关系。

导致多元共线性的因素有很多，较为常见的就是数据不足，这种还好，我们可以通过收集更多数据解决它。有一些本来就错的用法就只能靠避免了，例如一个模型的自变量有身高和体制，这两个变量虽然也不是确定的，但是关联性较大，就会导致多元共线性。或者举一个更笨蛋的例子，如果有一种物质，密度是确定的，质量公式为 $m = \rho v$ ，所以如果你把本来就成正比例关系的 m, ρ 在另外一个多元线性回归表达式作为两个自变量的话，它们之间存在确定的线性关系，就会导致更特殊的完全共线性（Perfect multicollinearity）。

因为涉及到一些其他概念，我们在后面会继续讨论如何处理多元共线性，先继续吧。

评估回归模型与优化

我们拟合出线性回归模型后，需要对其的预测能力和拟合程度进行评估，然后进行对应的优化。本章涉及的很多知识点都只能触及表面，将其深挖可能又是一篇文章，原谅我这次只能比较简单地表述它们。

损失函数

损失函数（Loss function）是一个衡量回归模型预测值与真实观测值误差的函数，它有很多种形式，应着这次主题介绍几种在回归模型比较常见的。接下来我们统一假设 y_i 是实际值， $\hat{y}_i$ 是预测值，则残差 $\varepsilon_i = y_i - \hat{y}_i$，但是某些地方为了更清晰就不代入$\varepsilon$了

均方误差

均方误差（Mean Squared Error）是一种比较适用于回归模型评估的损失函数，如名，我们通过计算残差平方的平均值获得：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

绝对平均值误差

我们通过通过计算残差绝对值的平均值获得绝对平均值误差（Mean Absolute Error），没有平方的话，差值有正有负，我们仍然需要使用绝对值让它们都是正数，避免正负误差相抵消。

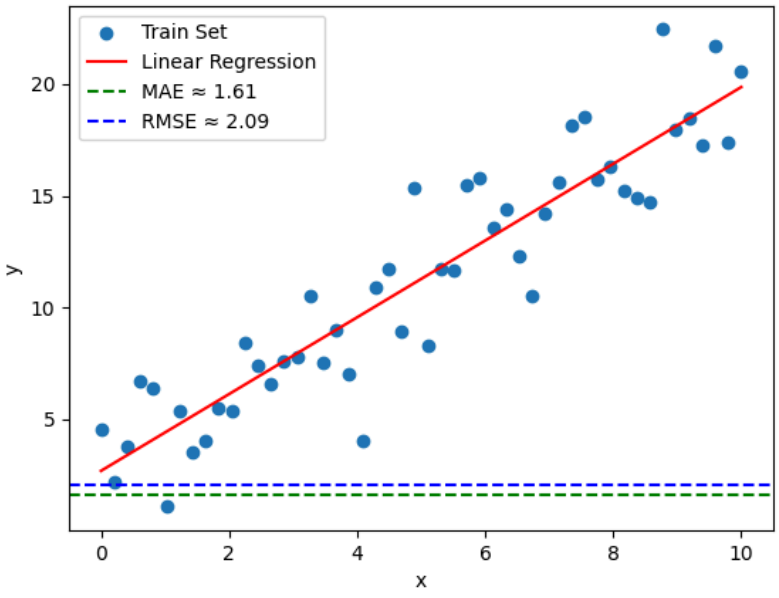
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

均方根误差

既然一个是残差平方一个是残差绝对值，你拿MSE和MAE直接比较很显然不公平，所以如果要比较，我们就应该在MSE的基础上开根号，这叫做均方根误差（Root Mean Square Error）

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

显然的是，平方使得RMSE对较大的误差会更敏感，相对应的MAE对于误差权重是一样的，我们应该根据实际情况选择使用的损失函数。大概是这样，图中可见RMSE比MAE的值要高，此外这个图的数据是随机生成的（拟合直线和误差值有计算），所以误差结果是很长的小数，此处保留小数点后二位。



R-squared

R-squared（决定系数）是一个衡量回归模型对观测数据的拟合程度，取值范围位于0-1之间，越接近0就是模型越无法解释因变量的变异性，也就是在数据集中观测值之间存在的差异或波动的程度（变异性你可以这样通俗的理解，再说严格一点就是数据点相对于期望值的离散程度），越接近1就表示模型越能解释因变量的变异性，也就是在范围内越高的R-squared就表示模型对数据的拟合程度越好。

若 S_r 是残差平方和（这里为了区别，加了下标），S_t 是总平方和，即因变量与其均值差的综合，则R-squared的计算公式为：

$$R^2 = 1 - \frac{Sr}{St} = 1 - \frac{(y_i - \hat{y}_i)^2}{(y_i - \bar{y})^2}$$

VIF评估

检测多元共线性的方法有很多种，在这里我介绍一种比较简单且常见的指标VIF（Variance Inflation Factor）。

我们在拟合完一个回归模型后，会得到对应的R-squared，此处表示对于第 i 个自变量与其他自变量之间的决定系数，将其作为因变量，使用其他自变量作为此处的自变量进行回归分析，计算回归模型的决定系数，所以VIF公式为：

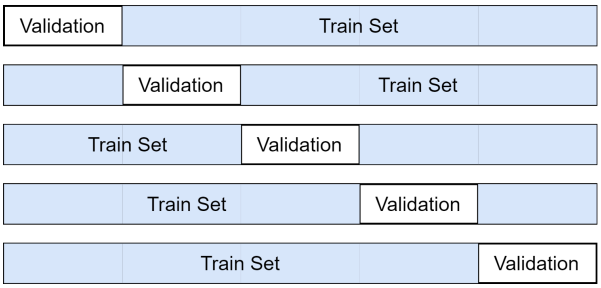
$$VIF = 1/(1 - R_i^2)$$

若VIF为1则代表对应的自变量间不存在多元共线性，5以上则是严重多元共线性了（标准不同，有些是VIF为10才当作严重共线性。很多时候多元共线性问题是无法完全避免的，我们能做的就是正确使用自变量，错误的使用自变量所导致的多元共线性是无法通过其他手段去除的，而如果是单纯样本不足的情况则可以通过收集足够的数据等手段来减小影响。

简单交叉验证

我们可以使用交叉验证（Cross-Validation）评估回归模型，大概的思想就是将数据集划分为训练集和验证集，反复训练和评估模型。交叉验证的方式有很多....如果要细讲可以另外写一篇Blog了，所以这里我们以最基本点 K-Fold交叉验证为例。

在K-Fold交叉验证中，数据集被分成 k 个护不重复的折，其中 $k-1$ 个作为训练集，其余的作为验证集，在每个折中使用训练集来训练机器学习模型并使用验证集评估模型，迭代完成后得到 k 个性能指标，求平均值作为最后的性能评估。



例如我们将数据分为5个折，则程序按照如图的顺序进行拟合和验证，最后求得到性能指标的平均值，这个计算很简单，选择一个合适的误差函数，并将计算后的误差 E 代入求平均值。

$$E = \frac{1}{n}\sum_{i=1}^{n} E_i$$

但是进行K-Fold交叉验证不可忽略的就是性能，若将数据集分为 n 折，则需要迭代、验证n次，其时间复杂度为 $O(n)$，而这也与具体的拟合和验证方式有关，如果模型很复杂，可能时间复杂度会很高。

所以除了这最基本的、最直观的交叉验证方式以外，还有很多更高级、更精准、更高效的方法可以使用，但是现在我们要开始考虑优化模型了。

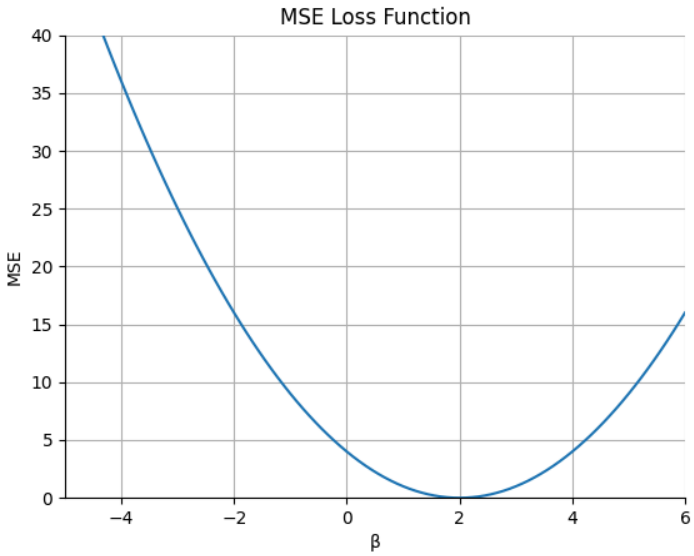
梯度下降算法

有了前面的评估指标，我们可以直接想到，优化线性回归模型需要使损失函数最小。梯度下降（Gradient descent）算法使用的是搜索的方法，选择一个初始点作为起点，开始不断搜索，损失函数逐渐变小，沿着图表函数梯度的负方向逐步接近最优解。这个知识点有点 复杂但比较重要，所以我们来详细点了解下（以后有空会专门写）

梯度是一个表示函数在给定点上沿着各方向变化率的向量，若函数有多个自变量，则梯度由各个偏导数组成，我们使用 ∇ 表示梯度：

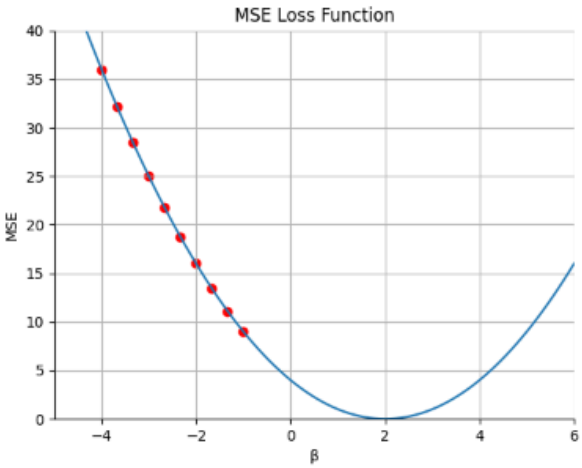
$$\nabla L(\beta) = \left(\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_2}, ..., \frac{\partial L}{\partial x_n} \right)$$

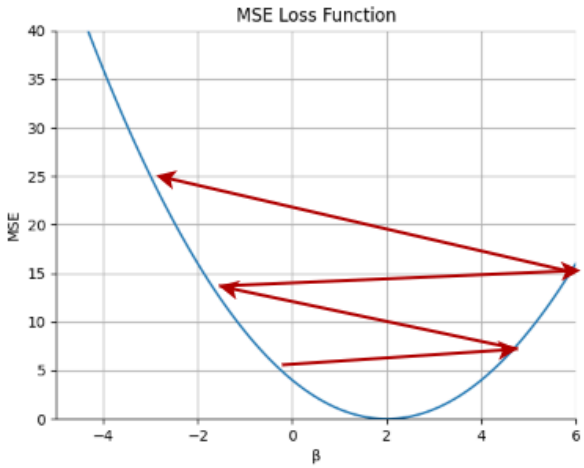
若 β 是模型中的回归系数，考虑均方误差损失函数 $L(\beta) = (\beta - 2)^2$ ，从图像看误差最小时 $\beta = 2$ ，则表明在最优情况下模型的参数应该接近2。我们沿着损失函数的负梯度方向更新参数以逐步接近最优解，若有学习率 α，则



$$\beta := \beta - \alpha \nabla L(\beta)$$

学习率 α 是一个超参数，它控制了我们计算损失函数关于参数的梯度的更新幅度，若这个值过大则会导致参数更新的步长过大，跳过最优解或者无法收敛，反之过小会导致算法过慢或者陷入局部最小值。你可以想象一个步长太大的取值在图像较窄的地方因为太大无法继续向下，就在两边反复横跳。如下图，左侧为学习率过小，右侧为学习率过大的情况





我们以 $L(\beta) = (\beta - 2)^2$ 为例，来尝试进行一个简单的推导，根据链式法则，我们可以先函数 $L(\beta)$ 看作两个函数的复合函数 $f(g(\beta))$，我们有

$$\frac{\partial L}{\partial \beta} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial \beta}$$

对于计算内部函数 $g(\beta)$ 关于 β 的导数 $\frac{\partial g}{\partial \beta} = \frac{\partial}{\partial \beta}(\beta - 2)$，常数项 -2 被置为0，求得 $\frac{\partial g}{\partial \beta} = 1$ 。外部函数 $f(u)$ 关于 u 的导数 $f(u) = u^2$，求导得到 $\frac{df}{du} = 2u$，我们将求导结果代回去：

$$\frac{\partial L}{\partial \beta} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial \beta} = 2u \cdot 1 = 2(\beta - 2)$$

将结果代回最开始的公式，得到

$$\beta := \beta - 2\alpha(\beta - 2)$$

要注意的是，在更新公式中，梯度符号决定了参数更新的方向，我们这里要沿着负梯度下降，所以是减号，若是正梯度则代表我们在当前点上升，若按照我们上述计算梯度的方式算得梯度为负，则参数会相应的增加（此处可以尝试自己写一个例子并推导）

基于这个最基本的梯度下降概念，还有很多梯度下降算法的拓展，什么Momentum、Nesterov Accelerated Gradient都很值得写，在以后的文章也会给各位介绍。

正则化方法简介

正则化是一种常用的优化方法，用于控制模型的复杂度并防止过拟合。正则化通过在损失函数中引入额外的惩罚项，鼓励模型选择较简单的解。简单来说，过拟合是指模型过于适应训练数据的误差和噪声，而忽略了真实的数据生成规律。这也是可以另外开一篇的话题，所以我们先在本文了解比较基础的两种正则化方法。

Lasso 正则化

假设有一个参数 λ 用来调节正则化的强度，越大则正则化强度越强，这个参数被我们叫做惩罚系数或者正则化系数，我们可以通过方式来选择这个参数，例如交叉验证的方式算出指标来作为参考。若损失函数为 $L(\beta)$，m 是自变量的数量，则在Lasso正则化中约束模型复杂性的正则项为

$$\lambda \sum_{j=1}^{m} |\beta_j|$$

Lasso对损失函数的形式没有特定要求，例如我们将均方误差的形式应用Lasso正则化，则应用后的损失函数为

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{m} |\beta_j|$$

岭回归

岭回归和Lasso类似，但是从定义上看，Lasso的惩罚项是模型参数的绝对值的和，而岭回归的惩罚项是模型参数的平方和。

$$\lambda \sum_{j=1}^{m} \beta_j^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{m} \beta_j^2$$

因为误差（这里应该叫噪声）是随机的，振幅较小，所以在岭回归中引入平方和作为惩罚项可以更有效地平滑模型的参数估计，减少噪声影响。

这两种正则化方式的选择重点在于场景，Lasso回归可能更适用于自变量较多且希望衡量其权重的情况，它更经常将系数置为0。相比之下的岭回归更适合用于优化具有多重共线性的情况，一般而言岭回归的结果是非0的，它会通过减少相关特征系数来减弱多重共线性的影响。

结尾

有点迫不得已的戛然而止，现在你已经了解了一些有关线性回归的基础概念。

这篇Blog写的稍微有点赶，急着在期末考前发，缺少了一些知识点和细节。但是我在写时就已经大概有哪里可以继续写哪里可以补充的想法，所以本片姑且当V1，我之后闲下来会补充细节和示例并发V2。另外，~~如果这篇文章有人看，~~我希望可以收到有关某个知识点可以专门写blog的建议。

顺便一提我发现ML好好玩，本来只是抱着试试看的心态学的，我近期开始写一个Haskell的机器学习库，之后如果有更新也能写博客分享。

主要参考书籍

Introduction to machine learning

MAKIROR gzanan@gmail.com 04:13 20/06/2023