第 2 章 从零认识线性回归

目录

第 2 章 线性回归.....1
2.1 模型的建立与求解.....1
2.1.1 理解线性回归模型.....1
2.1.2 建立线性回归模型.....2
2.1.3 求解线性回归模型.....2
2.1.4 sklearn 简介.....3
2.1.5 安装 sklearn 以及其它库.....3
2.1.6 线性回归示例代码.....4
2.1.7 小结.....5
2.2 多变量线性回归.....5
2.2.1 理解多变量.....5
2.2.2 多变量线性回归建模.....6
2.2.3 多变量回归示例代码.....6
2.3 多项式回归.....7
2.3.1 理解多项式.....7
2.3.2 多项式回归建模.....7
2.3.3 多项式回归示例代码.....7
2.3.4 小结.....8
2.4 回归模型评估.....9
2.4.1 常见回归评估指标.....9
2.4.2 回归指标示例代码.....11
2.4.3 小结.....11
2.5 梯度下降.....11
2.5.1 方向导数与梯度.....12
2.5.2 梯度下降算法.....12
2.5.3 小结.....15
2.6 正态分布.....15
2.6.1 一个问题的出现.....15
2.6.2 正态分布.....17
2.7 目标函数推导.....18
2.7.1 目标函数.....18
2.7.2 求解梯度.....19
2.7.3 矢量化计算.....20
2.7.4 从零实现线性回归.....21
2.7.5 小结.....22

第 2 章 线性回归

经过第 1 章的介绍，我们已经完成了对于 Python 开发环境的安装与配置。现在，就正式开始介绍第一个算法：线性回归（Linear Regression）。

整个线性回归的学习路线如图 2-1 所示。由于这是介绍的第一个算法，所以我们会介绍很多基本的内容，导致看起来有很多内容。因此，对于整个线性回归的学习将会通过 7 个小节的内容来进行介绍。但是，值得高兴的是只要完成了前 4 个步骤的学习就基本上是达到了第一阶段的要求。下面就将正式开始介绍线性回归。

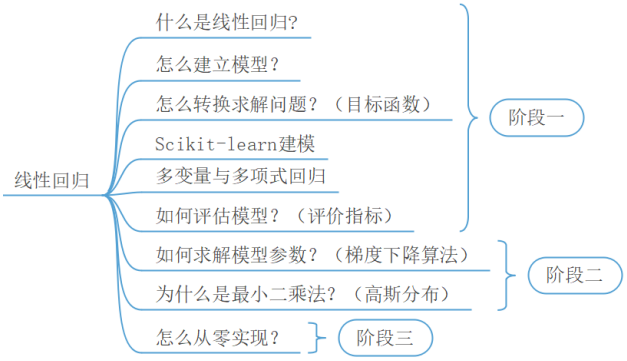


图 2-1 线性回归学习路线图

2.1 模型的建立与求解

2.1.1 理解线性回归模型

通常来说，机器学习中的每一个算法都是为了解决某一类问题而诞生。换句话说，也就是在实际情况中存在一些问题能够通过线性回归来解决，例如对房价的预测。但是有人可能会问，为什么对于房价的预测就应该用线性回归，而不是其它算法呢？其原因就在于常识告诉我们房价都是随着面积的增长而增长，且总体上呈线性增长的趋势。那有没有那种当面积大到一定程度后价格反而降低，因此不符合线性增长的呢？这当然也可能存在，但在实际处理中肯定会优先选择线性回归模型，当效果不佳时我们会再尝试其它算法。因此，当学习过多个算法后，在拿到某个具体的问题时，就需要考虑哪种模型更适合来解决这个问题了。

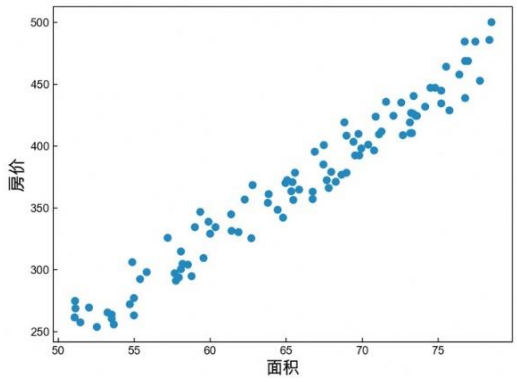


图 2-2 房价走势图

某市房价的一个走势图如图 2-2 所示，其中横坐标为面积，纵坐标为价格，且房价整体上呈线性增长的趋势。那假如现在随意告诉你一个房屋的面积，要怎么才能预测（或者叫计算）出其对应的价格呢？

2.1.2 建立线性回归模型

一般来说，当我们拿到一个实际问题时，首先会根据问题的背景结合常识选择一个合适的模型。同时，现在常识告诉我们房价的增长更优先符合线性回归这类模型，因此可以考虑建立一个如下的线性回归模型

$$\hat{y} = h(x) = wx + b \tag{2-1}$$

其中 w 叫权重参数（Weight），b 叫偏置（Bias）或者截距（Intercept）。当求解得到未知参数 w,b 之后，也就意味着我们得到了这个预测模型，即给定一个房屋面积 x，就能够预测出其对应的房价。

注意：在机器学习中所谓的模型，可以简单理解为一个函数。

2.1.3 求解线性回归模型

当建立好一个模型后，自然而然想到的就是如何通过给定的数据，也叫训练集（Training Data），来对模型 $h(x)$ 进行求解。在中学时期我们倒是学过如何通过两个坐标点来求解过这两点的直线，可在上述的场景中这种做法显然行不通的（因为所有的点并不在一条直线上），那有没有什么好的解决的办法呢？

此时就需要我们转换一下思路了，既然不能直接来进行求解那就换一种间接的方式。现在来想象一下，当 $h(x)$ 满足一个什么样的条件时，它才能称得上是一个好的 $h(x)$？回想一下求解 $h(x)$ 的目的是什么，不就是希望输入面积 x 后能够输出"准确"的房价 y 吗 ？既然直接求解 $h(x)$ 不好入手，那么我们就从“准确”来入手。

可又怎么来定义准确呢？在这里，我们可以通过计算每个样本的真实房价与预测房价之间的均方误差来对“准确”进行刻画

$$\begin{aligned} J(w,b) &= \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 \\ \hat{y}^{(i)} &= h(x^{(i)}) = wx^{(i)} + b \end{aligned} \tag{2-2}$$

其中，m 表示样本数数量；$x^{(i)}$ 表示第 i 个样本的，也就是第 i 个房屋的面积；$y^{(i)}$ 表示第 i 个房屋的真实价格；$\hat{y}^{(i)}$ 表示第 i 个房屋的预测价格。

由式(2-2)可知，当函数 $J(w,b)$ 取最小值时的参数 $\hat{w},\hat{b}$，就是要求的目标参数。为什么？因为当 $J(w,b)$ 取最小值就意味着此时所有样本的预测值与真实值之间的误差（Error）最小。如果极端一点，那就是所有预测值都等同于真实值，此时的 $J(w,b)$ 就是 0 了。

因此，对于如何求解模型 $h(x)$ 的问题就转换成了如何最小化函数 $J(w,b)$ 的问题。而 $J(w,b)$ 也有一个专门的术语叫目标函数（Objective Function）或者是代价函数（Cost Function）亦或是损失函数。

至此，我们离第一阶段的学习目标就只差一步之遥，那就是如何通过开源框架来进行建模求解，并进行预测。至于求解过程到底怎样如何进行的，那就是第二阶段的任务了，下面开始完成第一阶段的最后一步。

2.1.4 sklearn 简介

如图 2-3 所示，scikit-learn 简称 sklearn[①]。它是一个开源的机器学习框架，常用的机器学习算法都可以在里面找到，例如线性回归、逻辑回归、决策树等等。同时，其 Python 化的设计风格对于 Python 用户来说也十分友好与易于上手，并且每个算法都给了相应的示例以及 API 文档。

图 2-3 sklearn 介绍图

2.1.5 安装 sklearn 以及其它库

在第 1 章介我们已经绍了如何配置 Python 环境，下面就开始在第 1 章中所新建的 Python 虚拟环境里安装所需要的包（库）。

1) 打开终端

如果是在 Windows 环境中，则先点击“开始”，然后找到 Anaconda Prompt 并打开，最后再激活相应的 Python 虚拟环境即可。如果是在 Linux 环境中，则直接打开命令行终端，然后再激活相应的虚拟环境。

2) 安装 Python 包

为了完成整个线性回归的建模任务以及结果的可视化，需要安装 sklearn 和 matplotlib 这两个 Python 包。如果之前在配置环境的时候，替换了 pip 源地址为清华镜像，则下载的速度会更快，如图 2-4 所示。

```
(py36) test@VM-0-15-ubuntu:~$ pip install sklearn matplotlib
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: sklearn in ./miniconda3/envs/p
Collecting matplotlib
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/d2/4
79a200c6b559/matplotlib-3.3.3-cp36-cp36m-manylinux1_x86_64.wh
     |                                | 11.6 MB 137 kB/s
```

图 2-4 sklearn 安装示意图

由于在安装相应的 Python 包后 PyCharm 会建立新的包索引，所以需要等待几分钟，如图 2-5 所示。

[①] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

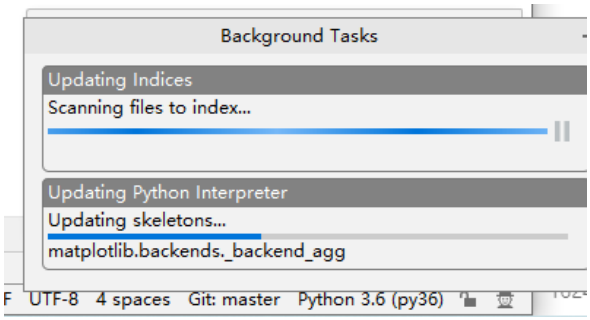


图 2-5 Python 包索引图

2.1.6 线性回归示例代码

在安装完成相应的 Python 包后就可以正式的对线性回归进行建模与求解，详细完整代码见 Chapter02/01_house_price_train.py 文件。

1) 导入包

首先需要将用到的相关 Python 包进行导入，代码如下：

```
1 import numpy as np
2 from sklearn.linear_model import LinearRegression
3 import matplotlib.pyplot as plt
```

2) 制作样本（数据集）

制作用于训练模型的数据集，代码如下：

```
1 def make_data():
2     np.random.seed(20)
3     x = np.random.rand(100) * 30 + 50  # 面积
4     noise = np.random.rand(100) * 50
5     y = x * 8 - 127 - noise              # 价格
6     return x,y
```

上述代码中第 2 行代码的作用是在真实的房价中加入一定的噪音（误差）。

3) 定义模型并求解

通过 LinearRegression 模型来对模型的参数进行求解与预测，代码如下：

```
1 def main(x, y):
2     model = LinearRegression()            # 定义模型
3     x = np.reshape(x, (-1, 1))
4     model.fit(np.reshape(x, (-1, 1)), y)# 求解模型
5     y_pre = model.predict(x) # 预测
6     print(y_pre)
```

上述代码中 np.reshape(x,(-1,1))表示把 x 变成[n,1]的形状，至于 n 到底是多少，将通过 np.reshape 函数自己推导得出。例如 x 的 shape 为[4,5]，如果想把 a 改成[2,10]形状则可以使用 a.reshape([2,10])，或者 a.reshape([2,-1])来进行形状的变换。

4) 运行结果

最后，调用定义好的函数运行程序，并输出最后训练得到的参数结果，代码如下：

```
if __name__ == '__main__':
    x, y = make_data()
    main(x, y)
    #参数 w=[7.97699647],b=-154.31885006061555
    #面积 50 的房价为： [244.53097351]
```

可以发现，其中参数 $w=7.97, b=-154.32$ 也就意味着 $h(x)=7.97x-154.32$。在这之后，便可以通过 $h(x)$ 来对新的输入进行预测。同时，还能够根据求解后的模型画出对应拟合出的直线，如图 2-6 所示。

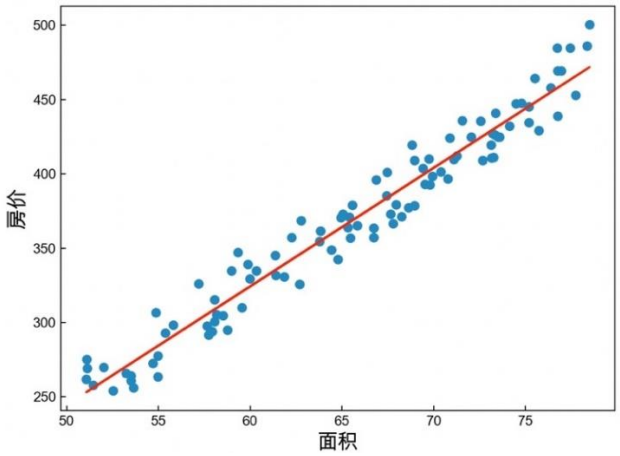


图 2-6 房价预测走势图

到此，便完成了对于线性回归第一阶段的大致学习。

2.1.7 小结

在本节内容中，笔者首先通过一个实际的场景介绍了什么是线性回归；接着介绍了如何建立一个简单的线性模型；然后引导大家如何将模型的求解问题，转化为目标函数的最小化过程；最后通过开源框架 sklearn 搭建了一个简单的线性回归模型并进行了求解。虽然内容不多，但是却体现了整个线性回归算法的核心思想。

2.2 多变量线性回归

2.2.1 理解多变量

在 2.1 节的内容中，笔者详细的介绍了什么是线性回归以及一个典型的应用场景；同时还介绍了如何通过开源的 sklearn 来搭建一个简单的线性回归模型，使得对于线性回归的核心思想有了一定的掌握；接下来，我们将继续开始学习线性回归后续的内容。

在这里还是以房价预测为例。尽管影响房价的主要因素是面积，但是其它因素同样也可能影响到房屋的价格。例如房屋到学校的距离、到医院的距离和到大型商场的距离等等（总不能卖你一套深山老林的房子你也要吧）。虽然现实生活中没有这么量化，但是开发商也总是会拿学区房做卖点。所以，这时便有了影响房价的四个因素，而在机器学习中我们将其称

之为特征（Feature）。因此，包含有多个特征的线性回归就叫做多变量线性回归（Multiple Linear Regression）。

2.2.2 多变量线性回归建模

以波士顿房价数据集为例，其一共包含了 13 个特征属性。因此可以得到如下线性模型

$$h(x) = w_1 x_1 + \cdots + w_{13} x_{13} + b \tag{2-3}$$

且同时，其目标函数为

$$\begin{aligned} J(W,b) &= \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 \\ \hat{y}^{(i)} &= h(x^{(i)}) = w_1 x_1^{(i)} + \cdots + w_{13} x_{13}^{(i)} + b \end{aligned} \tag{2-4}$$

其中 $x_j^{(i)}$ 表示第 i 个样本的第 j 个特征属性，W 为一个向量表示所有的权重，b 为一个标量表示偏置。

由 2.1 节的内容可知，只要通过某种方法最小化目标函数 $J(W,b)$ 后，便可以求解出模型对应的参数。

2.2.3 多变量回归示例代码

下面依旧以 sklearn 来进行多变量线性回归模型的建模与求解，完整代码见 Chapter02/02_boston_price_train.py 文件。

1) 导入数据集

这里直接导入了一个 sklearn 内置的 Boston 房价数据集为例进行演示，代码如下：

```
def load_data():
    data = load_boston()
    x = data.data
    y = data.target
    return x, y
```

2) 求解与结果

训练模型与输出相应的权重参数和预测值，代码如下：

```
def train(x, y):
    model = LinearRegression()
    model.fit(x,y)
    print("权重为：",model.coef_,"偏置为：",model.intercept_)
    print(f"第 12 个房屋真实价格为{y[12]},预测价格为
        {model.predict(x[12,:].reshape(1,-1))}")
```

根据上述代码便完成了对于多变量线性回归模型的建立与求解，同时也得出了各个特征所对应的权重参数。但由于不易对高维数据进行可视化，所以这里只能从预测的结果来评判模型的好坏，具体的模型评估指标将在第 2.4 节内容中进行介绍。

2.3 多项式回归

2.3.1 理解多项式

假定已知矩形的面积公式，而不知道求解梯形的面积公式，且同时你手上有若干个类似图 2-7 所示的梯形图。已知梯形的上底和下底，并且上底均等于高。现在让你建立一个模型，当给定一个上述梯形时你能近似的给出其面积。这该如何进行建模呢？

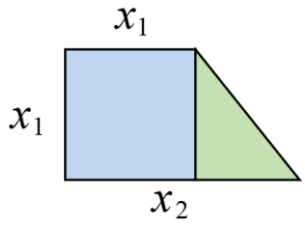


图 2-7 梯形图

2.3.2 多项式回归建模

首先需要明确的是，即使直接建模成类似 2.2 节中的多变量线性回归模型 $h(x)=w_1x_1+w_2x_2+b$ 也是可以的，只是效果不会太好。现在我们来分析一下，对于这个梯形，左边可以看成是正方形，所以可以人为的构造第三个特征 $(x_1)^2$；而整体也可以看成是长方形的一部分，则又可以人为的构造出 x_1x_2 这个特征；最后，整体还可以看成是大正方形的一部分，因此还可以构造出 $(x_2)^2$ 这个特征。故，我们便可以建立一个式(2-5)所示的模型

$$h(x)=x_1w_1+x_2w_2+(x_1)^2w_3+x_1x_2w_4+(x_2)^2w_5+b \tag{2-5}$$

此时有人可能会问，有的部分重复累加了，计算出来的面积岂不大于实际面积吗？这当然不会，因为每一项前面都有一个权重参数 w_i 做系数，只要这些权重有正有负，那就不会出现大于实际面积的情况。同时，可以发现 $h(x)$ 中包含了 $x_1x_2,(x_1)^2,(x_2)^2$ 这些项（其次数高于 1），因此将其称之为多项式回归（Polynomial Regression）。

但是，只要进行如下替换，便又回到了普通的线性回归：

$$h(x)=x_1w_1+x_2w_2+x_3w_3+x_4w_4+x_4w_5+b \tag{2-6}$$

其中 $x_3=(x_1)^2,x_4=x_1x_2,x_5=(x_2)^2$，只是在实际建模时先要将原始两个特征的数据转化为 5 个特征的数据；同时在正式进行预测时，输入给模型 $h(x)$ 的也将是包含 5 个特征的数据。

2.3.3 多项式回归示例代码

要完成整个多项式回归的建模，首先要对原始的特征进行转换，构造模型所需要的新特征；然后是构造相应的数据集；最后在进行模型的训练与预测。完整代码见 Chapter02/03_trapezoid_polynomial_train.py 文件。

1) 特征转化

这里首先介绍一下 sklearn 中的多项式变换模块 PolynomialFeatures，它的作用便是对每个特征进行指定的幂次变换，代码如下：

```
from sklearn.preprocessing import PolynomialFeatures
a = [[3, 4], [2, 3]]
model = PolynomialFeatures(degree=2,include_bias=False)
b = model.fit_transform(a)
print(b) #输出结果：[[ 3.  4.  9. 12. 16.] [ 2.  3.  4.  6.  9.]]
```

2) 构造数据集

构造训练模型时所需要用到的数据集，代码如下：

```
def make_data():
    np.random.seed(10)
    x1 = np.random.randint(5, 10, 50).reshape(50, 1)
    x2 = np. random.randint(10, 16, 50).reshape(50, 1)
    x,y = np.hstack((x1, x2)), 0.5 * (x1 + x2) * x1
    return x, y
```

根据上述代码便得到了一个 50 行 2 列的样本数据，其中第一列为上底，第二列为下底。np.hstack 的作用是将两个 50 行 1 列的向量合并成一个 50 行 2 列的矩阵。

3) 建模求解与结果

首先对原始特征进行多项式构造处理，然后进行模型的训练，代码如下：

```
def train(x, y):
    poly = PolynomialFeatures(degree=2, include_bias=False)
    x_mul = poly.fit_transform(x)
    model = LinearRegression()
    model.fit(x_mul, y)
    print("权重为：", model.coef_) #[[0.  0.  0.5  0.5  0]]
    print("偏置为：", model.intercept_) # [0.]
```

根据上述代码建模求解后，就能够预测得到上底为 5，下底为 8 的梯形面积为 32.5（真实面积也是 32.）。并且，根据求解得到的权重和偏置得

$$\begin{aligned} h(x) &= x_1 \cdot 0 + x_2 \cdot 0 + x_3 \cdot 0.5 + x_4 \cdot 0.5 + x_5 \cdot 0 + b \\ &= 0.5 \cdot (x_1)^2 + 0.5 \cdot x_1 \cdot x_2 \\ &= 0.5 \cdot x_1 (x_1 + x_2) \end{aligned} \tag{2-7}$$

可以发现，此时模型居然已经自己总结（学习）出了梯形的计算公式。

2.3.4 小结

在本节内容中，笔者首先以两个示例来分别介绍了多变量线性回归和多项式回归；然后通过 sklearn 来对模型进行了求解；最后和读者一起领略到了算法的魅力所在。在 2.4 节中，我们将开始对模型的评估进行介绍。

2.4 回归模型评估

在 2.1 到 2.3 这 3 节内容中，笔者介绍了如何建模线性回归（包括多变量与多项式回归）、如何通过 sklearn 搭建模型并求解。但是对于一个求解出来的模型应该怎样来对其进行评估呢？换句话说，这个模型到底怎么样？

以最开始的房价预测为例，现在假设你求解得到了图 2-8 所示的两个模型 $h_1(x)$ 与 $h_2(x)$，那么应该选哪一个呢？亦或是在不能可视化的情况下，应该来如何评估模型的好与坏呢？

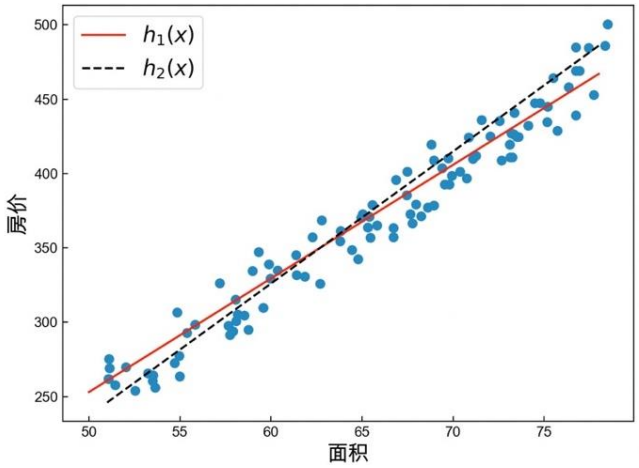


图 2-8 不同模型房价的预测走势图

在回归任务（对连续值的预测）中，常见的评估指标（Metric）有平均绝对误差（Mean Absolute Error,MAE）、均方误差（Mean Square Error,MSE）、均方根误差（Root Mean Square Error,RMSE）、平均绝对百分比误差（Mean Absolute Percentage Error,MAPE）和决定系数（Coefficient of Determination），其中用得最为广泛的就是 MAE 和 MSE。下面依次来进行大致的介绍，同时在所有的计算公式中，n 均表示样本数量、y_i 均表示第 i 个样本的真实值、$\hat{y}_i$ 均表示第 i 个样本的预测值。

2.4.1 常见回归评估指标

1) 平均绝对误差（MAE）

MAE 用来衡量预测值与真实值之间的平均绝对误差，定义如下：

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2-8}$$

其中 $MAE \in [0, +\infty)$，越小表示模型越好，实现代码如下所示：

```
1 def MAE(y, y_pre):
2     return np.mean(np.abs(y - y_pre))
```

2) 均方误差（MSE）

MSE 用来衡量预测值与真实值之间的误差平方，定义如下：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2-9}$$

其中 $MSE \in [0,+\infty)$，越小表示模型越好，实现代码如下所示：

```
def MSE(y, y_pre):
    return np.mean((y - y_pre) ** 2)
```

3) 均方根误差（RMSE）

RMSE 是在 MSE 的基础之上开根号而来，其定义如下：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2-10}$$

其中 $RMSE \in [0,+\infty)$，越小表示模型越好，实现代码如下所示：

```
def RMSE(y, y_pre):
    return np.sqrt(MSE(y, y_pre))
```

4) 平均绝对百分比误差（MAPE）

MAPE 和 MAE 类似，只是在 MAE 的基础上做了标准化处理，其定义如下：

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{2-11}$$

其中 $MAPE \in [0,+\infty)$，越小表示模型越好，实现代码如下所示：

```
def MAPE(y, y_pre):
    return np.mean(np.abs((y - y_pre) / y))
```

5) R^2 评价指标

决定系数 R^2 是线性回归模型中 sklearn 默认采用的评价指标，其定义如下：

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2-12}$$

其中 $R^2 \in (-\infty,1]$，越大表示模型越好，$\bar{y}$ 表示真实值的平均值，实现代码如下所示：

```
def R2(y, y_pre):
    u = np.sum((y - y_pre) ** 2)
    v = np.sum((y - np.mean(y_pre)) ** 2)
    return 1 - (u / v)
```

2.4.2 回归指标示例代码

有了这些评估指标后，在对模型训练时就可以选择其中的一些指标对模型的精度进行评估。这里以前面波士顿房价的预测结果为例进行示例，完整代码见 Chapter02/04_metrics_boston_price.py 文件。

```
def train(x, y):
    model = LinearRegression()
    model.fit(x, y)
    y_pre = model.predict(x)
    print("MAE:{}, MSE:{}".format(MAE(y, y_pre),MSE(y,y_pre)))
    # MAE: 3.27, MSE:21.89
```

2.4.3 小结

在本节中，笔者详细的介绍了如何评价一个回归模型的好坏，以及一些常用的评估指标和实现方式。最后，笔者还通过波士顿房价预测示例来展示了评价指标的用法。到此，对于线性回归模型在整个阶段一部分的内容就介绍完了。在 2.5 节的内容中，笔者将继续介绍如何通过梯度下降算法来求解目标函数，以及这个目标函数的由来。

2.5 梯度下降

在 2.1.3 节中，笔者不假思索的给出了线性回归的目标函数 $J(W,b)$，但并没有给出严格的数学定义。同时，在求解的过程中也是直接通过开源框架所实现的，也不知道其内部的真正原理。因此，在这一节内容中我们将会仔细地学习目标函数的求解过程以及最小二乘法。

根据前面的介绍可知，梯度下降算法的目的是最小化目标函数，也就是一个求解的工具。当目标函数取到（或接近）全局最小值时，我们也就求解得到了模型所对应的参数。不过那什么又是梯度下降（Gradient Descent）呢？如图 2-9 所示，假设有一个山谷，并且你此时处于位置 A 处，那么请问以什么样的方向（角度）往前跳，你才能最快的到达谷底 B 处呢?

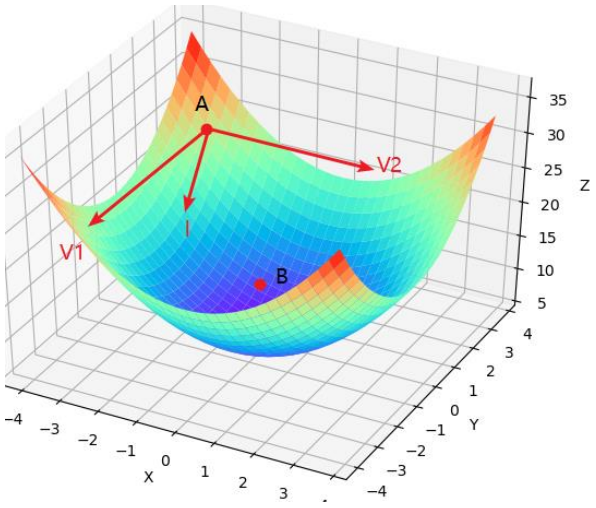


图 2-9 跳跃方向图

现在你大致有三个方向可以选择，沿着 Y 轴的 $\mathbf{V1}$ 方向，沿着 X 轴的 $\mathbf{V2}$ 方向以及沿着两者间的 $\mathbf{l}$ 方向。其实不用问，大家都会选择 $\mathbf{l}$ 所在的方向往前跳第一步，然后接着再选类似的方向往前跳第二步直到谷底。可为什么都会这样选呢？答：一看就知，不信你自己看。

2.5.1 方向导数与梯度

在学习一元函数导数的时候老师讲过，$f(x)$ 在 x_0 处的导数反映的是 $f(x)$ 在 $x=x_0$ 处时的变化率；$|f'(x_0)|$ 越大，也就意味着 $f(x)$ 在该处的变化率越大，即移动 Δx 后产生的函数增量 Δy 越大。同理，在二元函数 $z=f(x,y)$ 中，为了寻找 z 在 A 处的最大变化率，就应该计算函数 z 在该点的方向导数

$$\frac{\partial f}{\partial \mathbf{l}}=\{\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\}\cdot\{cos\alpha,cos\beta\}=|gradf|\cdot|\mathbf{l}|\cdot\cos\theta \tag{2-13}$$

其中 $\mathbf{l}$ 为单位向量，α,β 分别为 $\mathbf{l}$ 与 x 和 y 轴的夹角 θ 为梯度方向与 $\mathbf{l}$ 的夹角。

根据式(2-13)可知，要想方向导数取得最大值，那么 θ 必须为 0。由此可知，只有当某点处方向导数的方向与梯度的方向一致时，方向导数在该点才会取得最大的变化率。

在图 2-9 中，已知 $z=x^2+y^2+5$，A 的坐标为 $(-3,3,23)$，则 $\partial z/\partial x=2x,\partial z/\partial y=2y$。由此可知，此时在点 A 处梯度的方向为 $(-6,6)$。所以，当你站在在 A 点沿各个方向往前跳同样大小的距离时，只有沿着 $(\sqrt{2}/2,-\sqrt{2}/2)$ 这个方向（进行了单位化，且同时取了相反方向，因为这里要的是负增量）才会产生最大的函数增量 Δz。

如图 2-10 所示，要想每次都以最快的速度下降，则每次都必须向梯度的反方向向前跳。

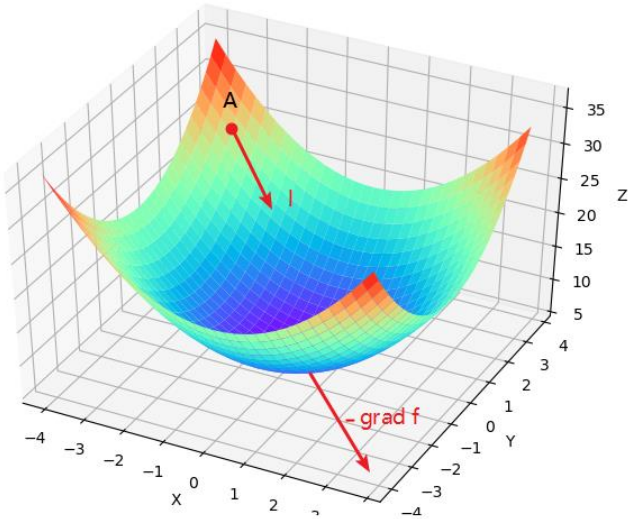


图 2-10 负梯度方向图

2.5.2 梯度下降算法

介绍这么多总算是把梯度这回事儿给说清楚了，那么如何用具体的数学表达式进行描述呢？总不能一个劲儿的喊它“跳”对吧。为了方便后面的表述以及将读者带入到一个真实求解的过程中，这里先将图 2-9 中的字母替换成模型中的参数表述。

现在有一个模型的目标函数 $J(w_1,w_2)=w_1^2+w_2^2+2w_2+5$（为了方便下面可视化此处省略了参数 b，但是原理都一样），其中 w_1,w_2 为待求解的权重参数，且随机初始化点 A 为初始权重值。下面就一步步的通过梯度下降法来进行求解。

如图 2-11 所示，设初始点 $A=(w_1,w_2)=(-2,3)$，则 $J(-2,3)=24$，且点 A 第一次往前跳的方向为 $-grad\ J=-(2w_1,2w_2+2)=(1,-2)$。

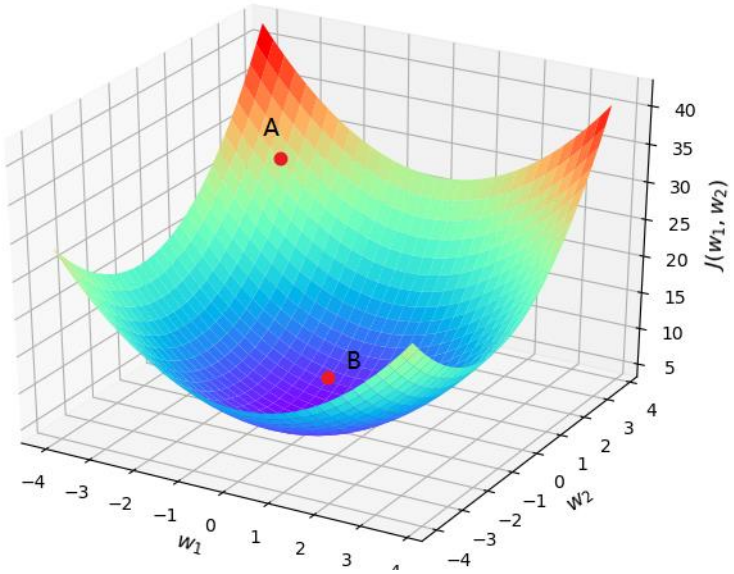


图 2-11 梯度下降图

如图 2-12 所示，**OQ** 为平面上梯度的反方向，**AP** 为其平移后的方向，但是长度为之前的 α 倍。因此，根据梯度下降的原则，此时曲面上的 A 点就该沿着其梯度的反方向跳跃，而投影到平面则为 A 应该该沿着 **AP** 的方向移动。假定曲面上 A 点跳跃到了 P 点，那么对应在投影平面上就是图 2-12 中的 **AP** 部分，同时权重参数也从 A 点更新到了 P 点的位置。

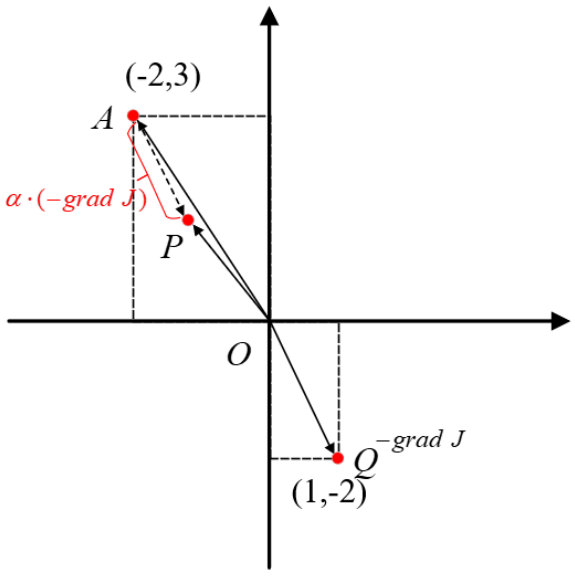


图 2-12 梯度计算图

从图 2-12 可以看出，向量 **AP**,**OA**,**OP** 三者的关系为

$$\mathbf{OP} = \mathbf{OA} - \mathbf{PA} \tag{2-14}$$

进一步，可以将(2-14)改写成

$$\mathbf{OP} = \mathbf{OA} - \alpha \cdot grad\ J \tag{2-15}$$

又由于$\mathbf{OP}, \mathbf{OA}$本质上就是权重参数w_1, w_2更新后与跟新前的值，所以便可以得出梯度下降的更新公式

$$W = W - \alpha \cdot \frac{\partial J}{\partial W} \tag{2-16}$$

其中$W = (w_1, w_2)$，$\partial J / \partial W$为权重的梯度方向，α为步长，用来放缩每次向前跳跃的距离。同时，将式(2-16)代入具体数值后可以得出，曲面上点A在第一次跳跃后的着落点为

$$\begin{aligned} w_1 &= w_1 - 0.1 \times 2 \times w_1 = -2 - 0.1 \times 2 \times (-2) = -1.6 \\ w_2 &= w_2 - 0.1 \times (2 \times w_2 + 2) = 3 - 0.1 \times (2 \times 3 + 2) = 2.2 \end{aligned} \tag{2-17}$$

此时，权重参数便从$(-2, 3)$更新到了$(-1.6, 2.2)$。当然其目标函数$J(w_1, w_2)$也从 24 更新到了 16.8。至此，我们便详细的完成了一轮梯度下降的计算。当A跳跃到P之后，又可以再次利用梯度下降算法进行跳跃，直到跳到谷底（或附近），如图 2-13 所示。

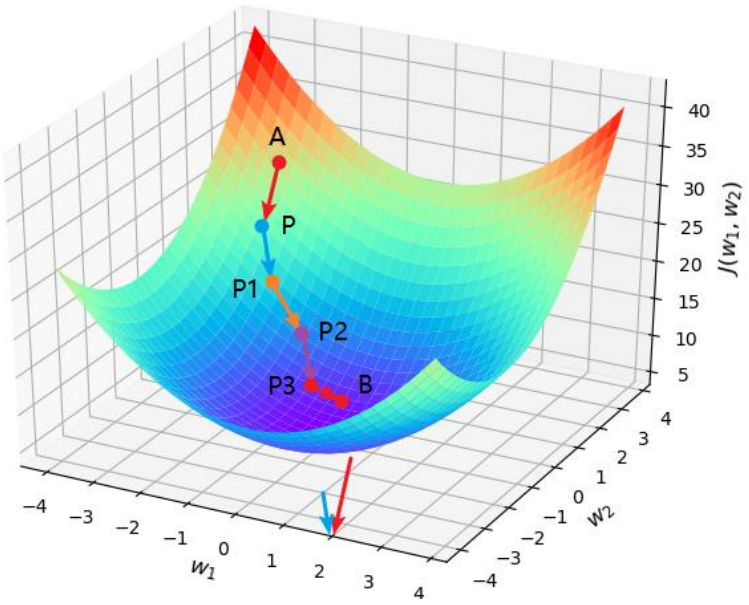


图 2-13 梯度下降示意图

最后，根据上述原理，还可以通过实际的代码将整个过程展示出来，完整代码见 Chapter02/05_gradient_descent_visualization.py 文件。

```
1 def gradient_descent():
2     w1, w2 = -2, 3
3     jump_points = [[w1, w2]]
4     costs, step = [cost_function(w1, w2)], 0.1
5     print("P:({,{)".format(w1, w2), end=' ')
6     for i in range(20):
```

```
        gradients = compute_gradient(w1, w2)
        w1 = w1 - step * gradients[0]    # 用来执行梯度下降过程
        w2 = w2 - step * gradients[1]
        jump_points.append([w1, w2])
        costs.append(cost_function(w1, w2))
        print("P{}:({},{})".format(i+1,
              round(w1,3), round(w2,3)), end=' ')
    return jump_points, costs
```

通过上述 Python 代码便可以详细展示跳向谷底时每一次的落脚点，并且可以看到谷底的位置就在 $(-0.023,-0.954)$ 附近，如图 2-14 所示。此致，就介绍完了如何通过编码来实现梯度下降算法的求解过程，等后续完成线性回归模型的推导后，再来自己编码完成线性回归模型的参数求解过程。

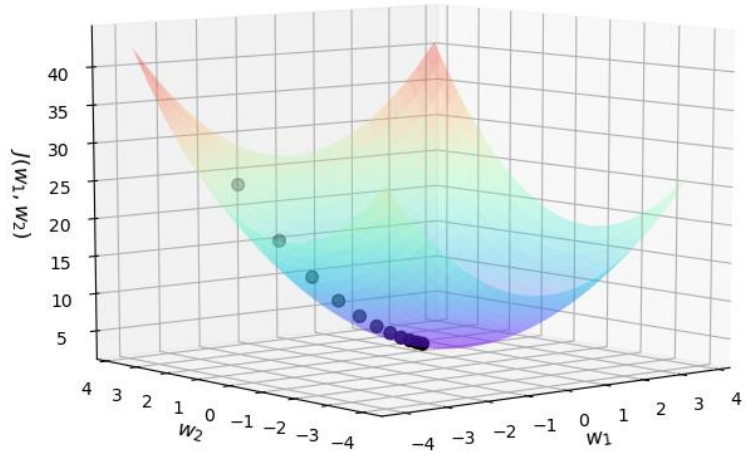


图 2-14 梯度下降可视化图

2.5.3 小结

在本节中，笔者通过一个跳跃的例子详细给大家介绍了什么是梯度，以及为什么要沿着梯度的反方向进行跳跃；然后通过图示导出了梯度下降的更新公式

$$W = W - \alpha \cdot \frac{\partial J}{\partial W} \tag{2-18}$$

在这里笔者又写了一遍是希望大家脑子里一定要记住这个公式，以及它的由来。因为它同时也是目前求解神经网络的主要工具。同时，可以看出，通过梯度下降算法来求解模型参数需要完成的一个核心就是计算参数的梯度。最后，虽然公式介绍完了，但公式中的步长 α 也是一个十分重要的参数，这将在第 3 章中进行介绍。

2.6 正态分布

2.6.1 一个问题的出现

17、18 世纪是科学发展的黄金年代，微积分的发展和牛顿万有引力定律的建立，直接的推动了天文学和测地学的迅猛发展。这些天文学和测地学的问题，无不涉及到数据的多次

测量、分析与计算。很多年以前，学者们就已经经验性的认为，对于有误差的测量数据，多次测量取算术平均是比较好的处理方法，并且这种做法现在我们依旧在使用。虽然当时缺乏理论上的论证，且也不断的受到一些人的质疑，但取算术平均作为一种直观的方式，被使用了千百年。 同时，算术平均在多年积累的数据处理经验中也得到相当程度的验证，被认为是一种良好的数据处理方法，但是在当时却没人能给出为什么。

1805 年，勒让德提出了一种方法来解决这个问题，其基本思想认为测量中存在误差，且让所有方程的累积误差为$\sum(\hat{y}-y)^2$ x，其中$\hat{y}$为观测值，y为理论值。然后通过最小化累积误差来计算得到理论值。即设真实值为θ，$x_1, x_2, ..., x_n$分别为n次独立观测后的测量值，每次测量的误差为$e_i = x_i - \theta$，按照勒让德提出的方法，累计误差为

$$E(\theta) = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (x_i - \theta)^2 \tag{2-19}$$

可以看出勒让德给出的方法其实就是最小二乘法（Least Square）。通过对$E(\theta)$求导后令其为 0，求解得到的结果正是算术平均$\bar{x} = 1/n \sum x_i$。由于算术平均是一个历经考验的方法，而以上的推理说明从另一个角度也说明了最小二乘法的优良性。这使得当时的人们对于最小二乘法有了更强的信心。从这里可以看出，这种做法的逻辑是，首先认为算术平均这种做法好但不知道为什么，然后有人提出了一种衡量误差的方法最小二乘，接着对误差最小化求解后发现其解正是算术平均，所以肯定了最小二乘的有用性。但事实上就是既没有说清楚算术平均为什么好，反而用算术平均的结果来肯定了最小二乘的作用。

与此同时，伽利略在他著名的《关于两个主要世界系统的对话》中也对误差的分布做过一些定性的描述。这主要包括①误差是对称分布的; ②大的误差出现频率低，小的误差出现频率高（这也很符合人们的认知常识）。用数学的语言描述，也就是说误差分布函数$f(x)$关于$x=0$对称分布，概率密度函数$f(x)$随$|x|$增大而减小，如图 2-15 所示。于是许多天文学家和数学家开始了寻找误差分布曲线的尝试，但最终都没能给出有用的结果。

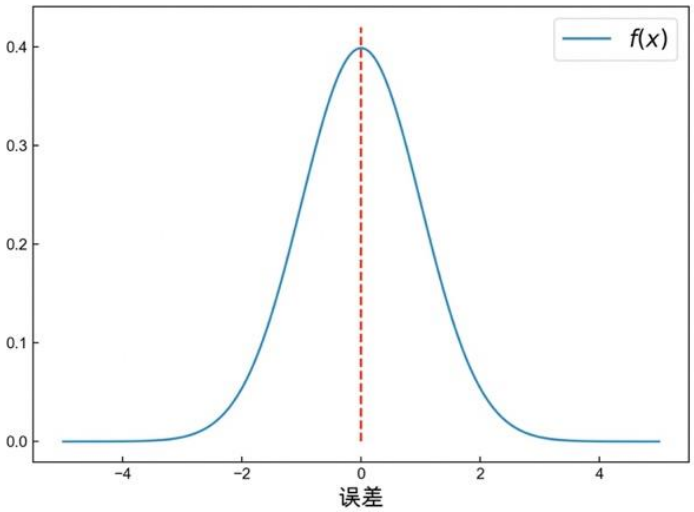


图 2-15 理想状态下误差分布图

2.6.2 正态分布

1801 年 1 月，天文学家朱塞普·皮亚齐发现了一颗从未见过的光度为 8 等的星在移动，这颗现在被称作谷神星（Ceres）的小行星在夜空中出现 6 个星期，扫过八度角后就在太阳的光芒下没了踪影无法观测。由于留下的观测数据有限难以计算出它的轨迹，所以天文学家们也因此无法确定这颗新星是彗星还是行星。不过这个问题很快成了学术界关注的焦点。高斯当时已经是很有名望的年轻数学家，这个问题引起了他的兴趣。高斯以其卓越的数学才能创立了一种崭新的行星轨道的计算方法，一个小时之内就计算出了谷神星的轨道，并预言了他在夜空中出现的时间和位置。1801 年 12 月 31 日夜，德国天文爱好者奥伯斯，在高斯预言的时间里，用望远镜对准了这片天空，果然不出所料，谷神星出现了。

高斯为此名声大震，但是高斯当时拒绝透露计算轨道的方法，原因可能是高斯认为自己的方法的理论基础还不够成熟。直到 1809 年高斯系统地完善了相关的数学理论后，才将他的方法公布于众，而其中使用的数据分析方法，就是以正态误差分布为基础的最小二乘法。那高斯是如何推导出误差分布为正态分布的呢？

设真实值为 θ， $x_1, x_2, ..., x_n$ 分别为 n 次独立观测后的测量值[①]，且每次测量的误差为 $e_i = x_i - \theta$。高斯假设误差 e_i 的密度函数为 $f(x)$，并直接把 n 个误差同时出现的概率记为

$$L(\theta) = L(\theta; x_1, x_2, ..., x_n) = f(e_1) f(e_2) \cdots f(e_n) \tag{2-20}$$

接着取使 $L(\theta)$ 达到最大值时的 $\hat{\theta}$ 作为 θ 的估计值，即使得式(2-21)成立时的 $\hat{\theta}$ 值。

$$L(\hat{\theta}) = \arg\max_{\theta} L(\theta) \tag{2-21}$$

现在我们把 $L(\theta)$ 称为样本的似然函数，而得到的估计值 $\hat{\theta}$ 称为 θ 的极大似然估计。在这里高斯首次给出了极大似然的思想，这个思想后来被统计学家费希尔系统的发展成为参数估计中的极大似然估计理论。同时，所谓最大似然估计是指在已知样本结果的情况下，推断出最有可能使得该结果出现的参数的过程。也就是说最大似然估计一个过程，它用来估计出某个模型的参数，而这些参数能使得已知样本的结果最可能发生。

接下来高斯把整个问题的思考模式倒了过来，既然千百年来大家都认为算术平均是一个好的估计，那我就认为极大似然估计导出的就应该是算术平均。所以高斯猜测上帝在创世纪中的旨意就是——误差分布导出的极大似然估计 = 算术平均值。然后高斯就开始去寻找满足这样条件的误差密度函数 $f(x)$，最后高斯应用数学技巧求解得到了这个函数 $f(x)$，并证明所有的概率密度函数中，唯一满足这个性质的就是

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{2-22}$$

其中 $\sigma > 0$ 为常数，而这也就是正态分布。

进一步，高斯基于这个误差密度函数对最小二乘法给出了一个漂亮的解释。对于最小二乘公式中涉及的每个误差 e_i，由式(2-20)可知其对应是似然估计为

[①] 靳志辉; 数学文化, 4 (2013), pp. 36-47.

$$L(\theta)=\prod_{i=1}^{n} f(e_i)=\frac{1}{(\sqrt{2\pi}\sigma)^n}\exp\left\{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}e_i^2\right\} \tag{2-23}$$

而要使得$L(\theta)$最大化，则必须使得$\sum_{i=1}^{n}e_i^2$取值最小，显然这正好就是最小二乘法的要求。可以看出，高斯这种做法的初始动机仍旧是以算术平均作为一种“公理”，然后以此为基础做出假设找到一种符合人们常识的误差密度函数，即正态分布。最后再通过最大似然估计来印证了最小二乘法。

2.7 目标函数推导

经过前面几节内容的介绍，我们知道了什么是线性回归、怎么转换求解问题、如何通过sklearn进行建模求解以及梯度下降法的原理与推导。同时，在2.6节中还通过一个故事来交代了最小二乘法的来历，以及误差服从高斯分布的事实。下面，我们就来完成线性回归中的最后两个任务，线性回归的推导以及Python代码的实现。

2.7.1 目标函数

根据前面的介绍，现在我们对线性回归的目标函数做如下定义：设样本为$(x^{(i)},y^{(i)})$，对样本的观测（预测）值记为$\hat{y}^{(i)}=W^Tx^{(i)}+b$，则有

$$y^{(i)}=\hat{y}^{(i)}+e^{(i)} \tag{2-24}$$

其中$e^{(i)}$表示第i个样本预测值与真实值之间的误差，W和$x^{(i)}$均为一个列向量，b为一个标量；同时由于误差$e^{(i)}$独立同分布于均值为0的高斯分布[①]，于是有

$$f(e^{(i)})=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(e^{(i)}-0)^2}{2\sigma^2}\right) \tag{2-25}$$

接着将(2-24)代入(2-25)有

$$f(e^{(i)})=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right) \tag{2-26}$$

此时请注意式(2-26)的右边部分（从右往左看），站在$y^{(i)}$的角度看显然是随机变量$y^{(i)}$服从以$\hat{y}^{(i)}$为均值的正态分布[②]（想想正态分布的表达式）。又由于此时的密度函数与参数$W,b,x^{(i)}$有关（即随机变量$y^{(i)}$是$x^{(i)},W,b$下的条件分布），于是有

$$p(y^{(i)}\mid x^{(i)};W,b)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right) \tag{2-27}$$

[①] Andrew Ng, Machine Learning, Stanford University, CS229, Spring 2019.

[②] Machine Learning - MT 2016 3. Maximum Likelihood

到目前为止，也就是说此时真实值 $y^{(i)}$ 服从均值为 $\hat{y}^{(i)}$，方差为 σ^2 的正态分布。同时，由于 $\hat{y}^{(i)}$ 是依赖于参数 W,b 的变量，那么什么样的一组参数 W,b 能够使得已知的真实值最容易发生呢？此时就要用到极大似然估计来进行参数估计

$$L(\theta)=\prod_{i=1}^{m}p(y^{(i)}\mid x^{(i)};W,b)=\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right) \tag{2-28}$$

为了便于求解，可以在等式(2-28)的两边同时取自然对数

$$\begin{aligned}\log L(W,b)&=\log\left\{\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right)\right\}\\&=\sum_{i=1}^{m}\log\left\{\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right)\right\}\\&=\sum_{i=1}^{m}\left\{\log\frac{1}{\sqrt{2\pi}\sigma}-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right\}\\&=m\cdot\log\frac{1}{\sqrt{2\pi}\sigma}-\frac{1}{\sigma^2}\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2\end{aligned} \tag{2-29}$$

由于 $\max L(W,b)$ 等价于 $\max\log L(W,b)$，所以

$$\begin{aligned}\max\log L(W,b)&\Leftrightarrow\min\frac{1}{\sigma^2}\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2\\&\Leftrightarrow\min\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2\end{aligned} \tag{2-30}$$

于是得目标函数

$$J(W,b)=\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2 \tag{2-31}$$

2.7.2 求解梯度

设 $y^{(i)}$ 表示第 i 个样本的真实值；$\hat{y}^{(i)}$ 表示第 i 个样本的预测值；W 表示权重（列）向量，W_j 表示其中一个分量；X 表示数据集，形状为 $m\times n$，m 为样本个数，n 为特征维度；$x^{(i)}$ 为一个（列）向量，表示第 i 个样本，$x_j^{(i)}$ 为第 j 维特征。

$$J(W,b)=\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2=\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)^2 \tag{2-32}$$

目标函数关于 W_j 的梯度求解过程为

$$\begin{aligned}\frac{\partial J}{\partial W_j}&=\frac{\partial}{\partial W_j}\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)}-(W_1x_1^{(i)}+W_2x_2^{(i)}\cdots W_nx_n^{(i)}+b)\right)^2\\&=\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-(W_1x_1^{(i)}+W_2x_2^{(i)}\cdots W_nx_n^{(i)}+b)\right)\cdot(-x_j^{(i)})\\&=\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)\cdot(-x_j^{(i)})\end{aligned}\tag{2-33}$$

目标函数关于b的梯度求解过程为

$$\begin{aligned}\frac{\partial J}{\partial b}&=\frac{\partial}{\partial b}\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)^2\\&=-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)\end{aligned}\tag{2-34}$$

此时便得到了目标函数关于参数的梯度计算公式

$$\begin{aligned}J(W,b)&=\frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)^2\\\frac{\partial J}{\partial W_j}&=\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)\cdot(-x_j^{(i)})\\\frac{\partial J}{\partial b}&=-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-(W^Tx^{(i)}+b)\right)\end{aligned}\tag{2-35}$$

在推导得到每个参数的梯度更新公式后，就能够根据梯度下降算法来迭代的对参数值进行更新，直到目标函数收敛。到此，对于整个线性回归部分阶段二的内容就介绍完了，下面继续来看最后一个阶段的内容。

2.7.3 矢量化计算

为了在编程时高效的计算，需要对式(2-35)进行矢量化，代码如下：

```
1 y_hat=np.matmul(X, W) + b
2 J(W,b)= 0.5 * (1 / m) * np.sum((y - y_hat) ** 2)
3 grad_w=-1/m*np.matmul(X.T,(y-y_hat))
4 grad_b=-1/m*np.sum(y-y_hat)
```

同时，这里有一个小技巧值得分享。当我们在矢量化公式的时，如果不知道哪个变量该放在哪个位置或者要不要进行转置，那么就带上变量的维度一起来进行计算。例如根据式(2-35)可以看出W_j 的梯度计算公式大致长成那样，所有矢量化后的结果也差不多是那个形式。同时W的形状是[n,1]，而真实值减去预测值后的形状为[m,1]，因此在和X计算后为了得W的形状，那么只能是一个[n,m]的矩阵乘以[m,1]的矩阵才可能的那样的结果。故，应该把X的转置放在前面。

2.7.4 从零实现线性回归

下面依旧以波士顿房价预测模型为例，然后通过手动编写代码进行模型的建模与求解，完整代码见 Chapter02/06_boston_price_train.py 文件。

1) 定义预测函数

首先需要定义一个预测函数，也就是 2.2 节内容中介绍的 $y = w_1x_1 + w_2x_2 \cdots w_nx_n + b$。为了同时对输入的所有样本进行计算，一般以两个矩阵相乘的方式进行，代码如下：

```
def prediction(X, W, bias):#预测
    return np.matmul(X, W) + bias  # [m,n] @ [n,1] = [m,1]
```

2) 定义目标函数

为了方便在训练结束后（或者训练时）观察目标函数的收敛情况，所以通常会计算每一次参数更新后的损失值，代码如下：

```
def cost_function(X, y, W, bias):  # 代价函数
    m, n = X.shape
    y_hat = prediction(X, W, bias)
    return 0.5 * (1 / m) * np.sum((y - y_hat) ** 2)
```

3) 定义梯度下降

在这里，需要完成模型训练时的核心部分，也就是整个梯度下降的过程，代码如下：

```
def gradient_descent(X, y, W, bias, alpha):
    m, n = X.shape
    y_hat = prediction(X, W, bias)
    grad_w = -(1 / m) * np.matmul(X.T, (y - y_hat))
    grad_b = -(1 / m) * np.sum(y - y_hat)
    W = W - alpha * grad_w
    bias = bias - alpha * grad_b
    return W, bias
```

可以看到，对于梯度求解就是(2-35)矢量化后的形式，再用梯度下降对参数更新即可。

4) 训练模型

在定义完训练过程中需要的相关函数后，接着就可以初始化相关权重和变量通过梯度下降算法来进行参数的更新，代码如下：

```
def train(X, y, ite=200):
    m, n = X.shape  # 506,13
    W, b, alpha, costs = np.random.randn(n, 1),0.1,0.2
    for i in range(ite):
        costs.append(cost_function(X, y, W, b))
        W, b = gradient_descent(X, y, W, b, alpha)
    y_pre = prediction(X, W, b)
    return costs
```

最后，可以通过如下所示的过程来完成整个模型的训练，代码如下：

```
if __name__ == '__main__':
    x, y = load_data()
    train_by_sklearn(x, y)
    costs = train(x, y)
    #输出结果：
    #MSE:  21.894831181729206
    #MSE:  21.901070160392404
```

同时，在这里也加入了通过 sklearn 训练后的模型的 MSE 评估值。可以看出，我们自己实现的模型与 sklearn 中的线性回归模型在 MSE 上几乎没有任何差别。大约在 25 次迭代后目标函数就开始进入收敛状态，如图 2-16 所示。

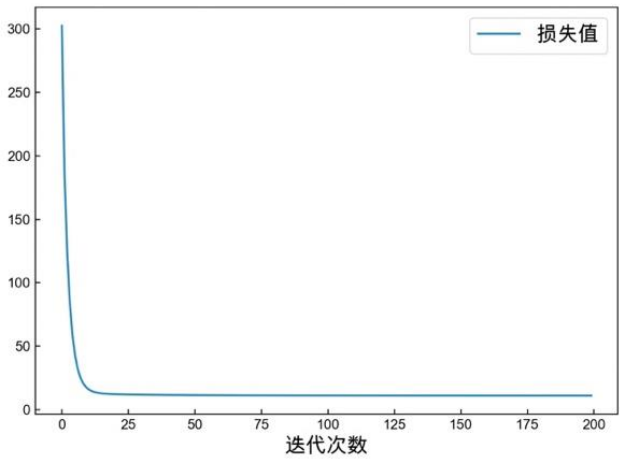


图 2-16 目标函数收敛图

2.7.5 小结

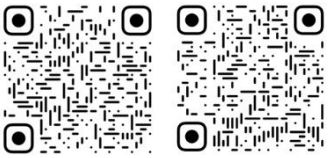
通过本节内容的学习，对于线性回归的主要内容也算到此结束了。对于其它细枝末节的地方（例如学习率的选择、特征标准化等），在后续模型改善部分再进行介绍。

图 2-17 学习层次图

总结一下，如图 2-17 所示，在本章中笔者首先介绍了线性回归模型第一阶段的主要内容，至此你可以大致知道拿到一个问题如何通过开源的 sklearn 库进行线性回归建模。虽然阶段一的内容并没有从数学的角度来完成对于线性回归的论证，但是其包含了学习一个算法并快速入门的路线。然后笔者接着介绍了梯度下降算法的介绍、误差的正态分布特性和目标函数的推导，完成了线性回归第二阶段的学习。这个阶段一般都是算法从数学层面上的理论

依据，尤其是统计机器学习更是依赖于这个过程，但是这部分通常来说都有一定的难度，例如 SVM 的求解过程。最后，笔者通过本节的内容完成了线性回归模型源码的实现。

本次内容就到此结束，感谢您的阅读！如果你觉得上述内容对你有所帮助，欢迎分享至一位你的朋友！若有任何疑问与建议，请添加笔者微信'nulls8'或加群进行交流。**青山不改，绿水长流，我们月来客栈见！**



扫码关注月来客栈可获得更多优质内容！

代码仓库： https://github.com/moon-hotel/MachineLearningWithMe

2021年

第一章：机器学习环境安装

Python版本为3.6，各个Python包版本见`requirements.txt`，使用如下命令即可安装：

```
pip install -r requirements.txt
```

第二章：从零认识线性回归　　代码

第三章：从零认识逻辑回归　　代码

第四章：模型的改善与泛化

第五章：K近邻算法

第六章：朴素贝叶斯算法

第七章：文本特征提取与模型复用

第八章：决策树与集成模型

第九章：支持向量机

第十章：聚类算法