Python程序设计（混合式）

py for AI

杨尚东
南京邮电大学计算机学院，数据科学与工程系
shangdongyang.github.io

2023/11/17

目录

离散vs连续

鸭嘴兽体表有毛，**用乳汁哺乳后代**，具有哺乳动物的特征；但鸭嘴兽的**繁殖方式是卵生**，又像爬行动物。

- 离散化引入了不可觉察的误差来抵御外部的干扰。
- 离散化简化了事务的描述方式，可以用简单的加减取代复杂的运算。
- 离散化可以描述更多更复杂的用连续性无法描述的事务。

分类问题简介

- 分类问题的预测值是离散的
  - ✓ 根据晚风和晚霞预测明天是否晴天
  - ✓ 根据户型、面积、价格等因素预测房子是否好卖
  - ✓ 根据气色、打喷嚏、食欲等估算是否感冒
  - ✓ 根据西瓜的外观、敲瓜响声判断西瓜是否甜
  - ✓ 根据餐馆的飘香、入座情况等判断菜品是否好吃

- 分类对人类来说是一个基本能力

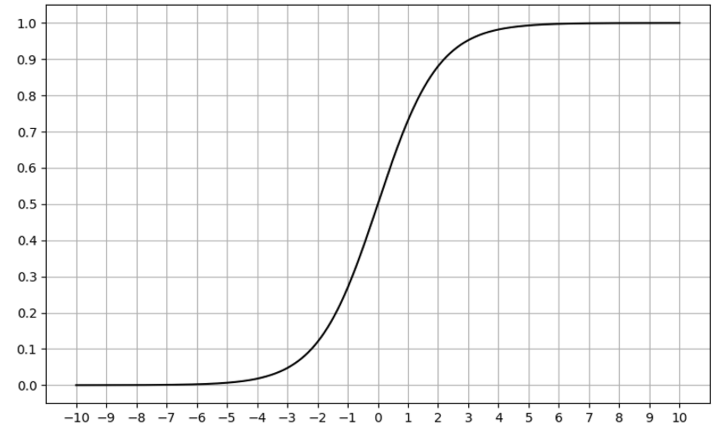
- 让人工智能学习分类是一个复杂的过程，需要**优秀的模型**、**海量的数据**和**高性能的硬件**支持

逻辑斯蒂分类简介

- 逻辑斯蒂分类（回归）（Logistic Regression），是一个回归方法，但通常用于**二分类**，也称为对数几率回归，逻辑回归

- 为了实现二分类，理想情况应该是一个**单位阶跃函数**
- 逻辑斯蒂分类通过拟合一个特殊的函数，即**逻辑斯蒂函数**（Logistic Function）进行分类

$$f(x)=\frac{1}{1+e^{-x}}$$



- $f(x)$值的取值范围在0~1之间
- 对于二分类问题，两个类分别用0和1表示
- 给定有 d 个属性的样例 $x=(x_1;\, x_2;\, x_3;\, \ldots;\, x_d)$

$$P(y=1\,|\,\mathbf{x})=\frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}} \qquad P(y=0\,|\,\mathbf{x})=\frac{e^{-(\mathbf{w}^T\mathbf{x}+b)}}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

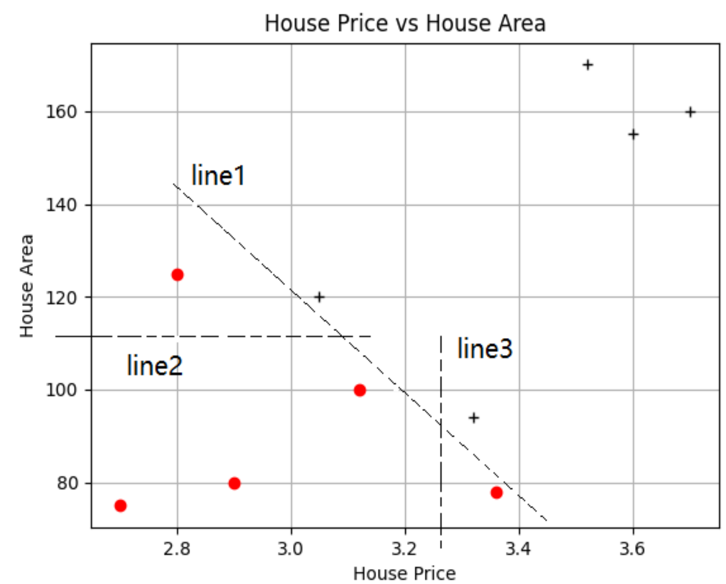
二分类逻辑斯蒂分类问题

- 逻辑斯蒂**分类类别数量只有两个**（即y的取值是0或1）
- 案例描述：
  - ✓ 根据历史销售数据，该小区有些房屋好卖（在挂售半年内就可以成交），有些房屋不好卖（在挂售半年后还不能成交），观察发现，房屋是否好卖跟房屋挂售的房屋面积和每平方米的单价有很大关系。
  - ✓ 下表例举了15条历史销售记录，包括10条训练样本和5条测试样本。现有该小区的一位业主出售房屋，在业主报出房屋面积和期望售价后，根据表中的训练数据，中介要判断该房屋是否好卖。

样本	房屋面积	房屋单价（万元/平米）	是否好卖
训练样本1	78	3.36	是
训练样本2	75	2.70	是
训练样本3	80	2.90	是
训练样本4	100	3.12	是
训练样本5	125	2.80	是
训练样本6	94	3.32	否
训练样本7	120	3.05	否
训练样本8	160	3.70	否
训练样本9	170	3.52	否
训练样本10	155	3.60	否
验证样本1	100	3.00	是
验证样本2	93	3.25	否
验证样本3	163	3.63	是
验证样本4	120	2.82	是
验证样本5	89	3.37	是

案例分析

```
#代码4.1 表4.1中房屋销售数据的可视化展示代码
import numpy as np
import matplotlib.pyplot as plt
def initPlot():
    plt.figure()
    plt.title('House Price vs House Area')
    plt.xlabel('House Price') #x轴标签文字
    plt.ylabel('House Area') #y轴标签文字
    plt.grid(True) #显示网格
    return plt
xTrain0 = np.array([[3.32, 94], [3.05, 120], [3.70, 160], [3.52, 170], [3.60, 155]]) #标注为不好卖的样本
yTrain0 = np.array([0, 0, 0, 0, 0]) #y=0表示不好卖
xTrain1 = np.array([[3.36, 78], [2.70, 75], [2.90, 80], [3.12, 100], [2.80, 125]]) #标注为好卖的样本
yTrain1 = np.array([ 1, 1, 1, 1, 1]) #y=1表示好卖
plt = initPlot()
plt.plot(xTrain0[:, 0], xTrain0[:, 1], 'k+') #k表示黑色，+表示点的形状为十字
plt.plot(xTrain1[:, 0], xTrain1[:, 1], 'ro') #r表示红色，o表示点的形状为圆形
plt.show()
```



[1] https://matplotlib.org/stable/gallery/index

LogisiticRegression类

- 使用Scikit-learn库的LogisticRegression类解决逻辑斯蒂分类问题
- **from sklearn.linear_model import LogisiticRegression**
- model=LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)
  - ✓ penalty：正则化参数，可选值为“L1”和“L2”
  - ✓ solver：优化算法选择参数
    - liblinear：使用坐标轴下降法来迭代优化损失函数
    - lbfgs：拟牛顿法的一种
    - newton-cg：也是牛顿法家族的一种
    - sag：随机平均梯度下降

LogisiticRegression类

- 使用Scikit-learn库的LogisticRegression类解决逻辑斯蒂分类问题
- **from sklearn.linear_model import LogisiticRegression**
- model=LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)
  - ✓ multi_class：分类方式选择参数
  - ✓ class_weight：类别权重参数
  - ✓ fit_intercept：是否存在截距
  - ✓ max_iter：算法收敛的最大迭代次数
- 拟合函数 **fit(X, y)**、预测函数 **predict(X)**、预测概率函数 predict_proba(X), 评价分数值 score(X, y)

求解步骤

- 第一步：准备训练数据
  - ✓ xTrain = np.array([[94], [120], [160], [170], [155], [78], [75], [80], [100], [125]])
  - ✓ yTrain = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
- 第二步：创建LogisticRegression对象并拟合
  - ✓ from sklearn.linear_model import LogisticRegression #导入类
  - ✓ model = LogisticRegression(solver = "lbfgs") #创建对象，默认优化算法是L-BFGS
- 第三步：执行拟合
  - ✓ model.fit(xTrain, yTrain) #执行拟合
  - ✓ print(model.intercept_) #输出截距
  - ✓ print(model.coef_) #输出斜率
- 第四步：对新数据执行预测
  - ✓ newX = np.array([[100], [130]]) #定义新样本
  - ✓ newY = print(model.predict(newX)) #输出斜率

代码4.2

```
#代码 4.2 基于 Scikit-learn 库求解房屋好卖预测问题
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

xTrain = np.array([[94], [120], [160], [170], [155], [78],
[75], [80], [100], [125]])
yTrain = np.array([ 0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
model = LogisticRegression(solver = "lbfgs")
model.fit(xTrain, yTrain)
newX = np.array([[100], [130]])
newY = model.predict(newX)

def initPlot():
    plt.figure()
    plt.title('House Price vs Is Easy To Sell')
    plt.xlabel('House Price')
    plt.ylabel('Is Easy To Sell')
    plt.grid(True)
    return plt
```

```
plt = initPlot()
plt.plot(xTrain[:5,0], yTrain[:5], 'k+')
plt.plot(xTrain[5:,0], yTrain[5:], 'ro')
print ("model.coef_： ", model.coef_ )
print ("model.intercept_： ", model.intercept_ )

#计算分割的中间 x 值
split_x = -model.intercept_[0] / model.coef_[0][0]
print("分割的中间 x 值：%.2f" % split_x)

x = np.linspace(30, 200, 10000)
y = 1 / (1 + np.exp(-(x * np.reshape(model.coef_, [-1])[0] +
model.intercept_[0])))

plt.plot(x, y, 'g-')  #格式字符串'g-'，表示绘制绿色的线段
#绘制坐标
plt.plot([split_x]*2, [0, 1], 'b-')
plt.text(split_x, 0.5, "({:.2f}, {})".format(split_x, 0.5))
plt.show()
```

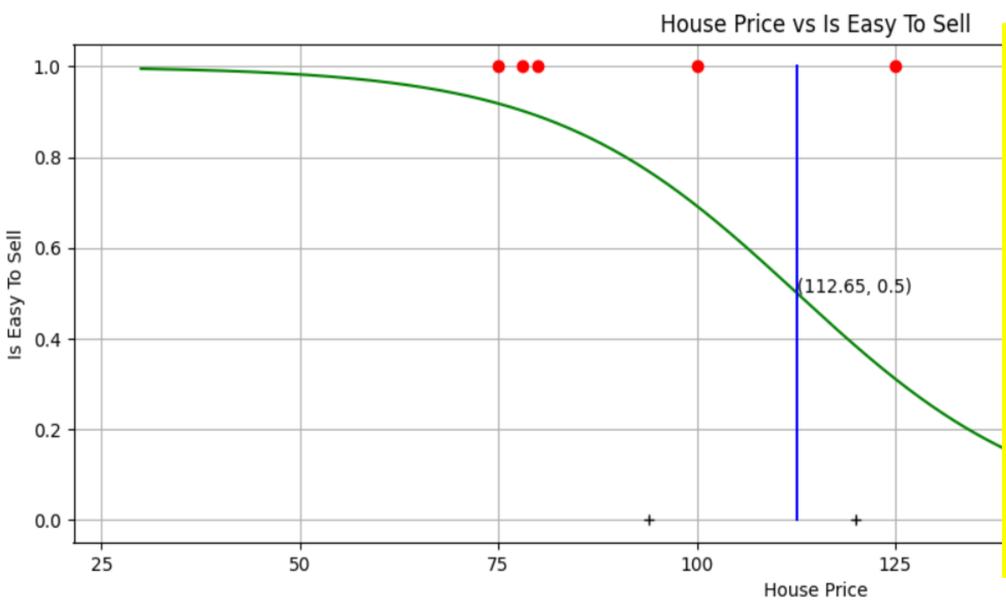
代码4.2

- 运行演示代码4.2

$$P(y=1|x)=f(x)=\frac{1}{1+e^{-(0.06426704x+7.23982418)}}$$

- 拟合得到函数：
  - ✓ 当x=112.65时，分母中的指数部分为零

$$P(y=1|x=112.65)=0.5$$



- 训练数据中有一个标记为“好卖”的样本（图中最右边的圆点）被分类函数错误地分类为“不好卖”（概率小于0.5，位于分割线的右边）
- 有一个标记为“不好卖”的样本（图中最左边的十字点）被分类函数错误地分类为“好卖”（概率大于0.5，位于分割线的左边）。

极大似然估计（Maximum Likelihood Estimation）

- 极大似然估计是一种用于统计推断的方法，用于根据观测到的数据来估计模型的参数值。在极大似然估计中，我们试图寻找**最大化观测数据概率的参数值**，以便这些参数值能够产生观测到的数据。

- 在极大似然估计中，我们假设观测到的数据是从某个概率分布中随机抽样得到的，并且我们知道该概率分布的函数形式，但不知道它的参数值。我们的目标是根据观测到的数据，找到最有可能生成这些数据的参数值。

- 举个例子，假设我们有一个硬币，我们不知道这个硬币正面朝上的概率是多少。我们可以进行一系列抛硬币实验，并记录每次抛硬币时硬币正面朝上的情况。如果我们假设这个硬币正面朝上的概率是 p，那么我们可以计算出每次实验中硬币正面朝上的概率，然后将所有实验的结果乘起来，得到观测到这些数据的概率。我们的目标就是找到一个最优的 p 值，使得这个概率最大化，从而能够最好地解释我们的数据。

梯度下降法优化目标

- 逻辑斯蒂分类的判别函数

$$P(y=1\mid \mathbf{x})=f(\mathbf{x})=\frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}}=\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

  - ✓ 其中:

$$\mathbf{x}^T=[1,x_1,x_2,\ldots x_d] \qquad \mathbf{w}^T=[w_0,w_1,w_2,\ldots w_d]$$

- 训练数据中有 m 个样本， $y^{(i)}=0$ 表示第 i 个样本的实际类别为第 0 类， $y^{(i)}=1$ 表示该样本的实际类别为第1类

- M_0 为实际类别为 0 的样本子集， M_1 为实际类别为 1 的样本子集

梯度下降法优化目标

- 对于一个 M_0 中的样本 i ，其预测概率为

$$P\left(y=0\middle|\mathbf{x}^{(i)}\right)=1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}$$

✓ **要尽量使得所有样本预测概率最大化**，这些样本满足独立同分布的性质；通常对这个函数取对数后进行优化

$$\max\frac{1}{|M_0|}\sum_{i\in M_0}\log\left(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}\right)$$

梯度下降法优化目标

- 对于一个 M_1 中的样本 i，其预测概率为

$$P\left(y=1 \middle| \mathbf{x}^{(i)}\right)=\frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}^{(i)}}}$$

  - ✓ **要尽量使得所有样本预测概率最大化**，这些样本满足独立同分布的性质；同样地，取对数后可得到

$$\max \frac{1}{|M_1|} \sum_{i \in M_1} \log\left(\frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}^{(i)}}}\right)$$

梯度下降法优化目标

- 将以上两类样本的优化目标合并之后，可以得到总的优化目标如公式

$$\max imize \frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}) + (1-y^{(i)}) \log(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})$$

- 左半部分是用实际类别为 1 的训练样本进行优化，左半部分是用实际类别为 0 的训练样本进行优化

- 优化目标一般是进行最小化而不是最大化，$L(\mathbf{w})$ 也被称为损失函数（Loss Function）

$$L(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \log(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}) - (1-y^{(i)}) \log(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})$$

$$\min L(\mathbf{w})$$

梯度计算

- 梯度下降法需要根据梯度更新参数，更新公式如下

$$w_j = w_j - \alpha * \frac{\partial L(\mathbf{w})}{\partial w_j}$$

- 偏导数的求解如下

$$f(\mathbf{x}^{(i)}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}$$

$$\frac{\partial L(\mathbf{w})}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \frac{1}{f(\mathbf{x}(i))} \bullet \frac{\partial f(\mathbf{x}^{(i)})}{\partial \mathbf{w}_j} - (1-y^{(i)}) \frac{1}{1-f(\mathbf{x}^{(i)})} \bullet \frac{-\partial f(\mathbf{x}^{(i)})}{\partial w_j}$$

$$\frac{\partial L(\mathbf{w})}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)} + \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}) \bullet x_j^{(i)}$$

梯度计算

- 推导过程

设： $f(\mathbf{x}^{(i)})=\dfrac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}$

则有： $\dfrac{\partial L(\mathbf{w})}{\partial w_j}=\dfrac{1}{m}\sum_{i=1}^{m}-y^{(i)}\dfrac{1}{f(\mathbf{x}(i))}\bullet\dfrac{\partial f(\mathbf{x}^{(i)})}{\partial \mathbf{w}_j}-(1-y^{(i)})\dfrac{1}{1-f(\mathbf{x}^{(i)})}\bullet\dfrac{-\partial f(\mathbf{x}^{(i)})}{\partial w_j}$

由于： $\dfrac{\partial f(\mathbf{x}^{(i)})}{\partial w_j}=-\dfrac{1}{(1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}})^2}\bullet(-x_j^{(i)}e^{-\mathbf{w}^T\mathbf{x}^{(i)}})$

$$=\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}\bullet\frac{e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}\bullet x_j^{(i)}$$

$$=f(\mathbf{x}^{(i)})\bullet(1-f(\mathbf{x}^{(i)}))\bullet x_j^{(i)}$$

代入后得： $\dfrac{\partial L(\mathbf{w})}{\partial w_j}=\dfrac{1}{m}\sum_{i=1}^{m}-y^{(i)}(1-f(\mathbf{x}^{(i)}))\bullet x_j^{(i)}-(1-y^{(i)})(-f(\mathbf{x}^{(i)})\bullet x_j^{(i)}$

$$=\frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}+f(\mathbf{x}^{(i)}))\bullet x_j^{(i)}$$

$$\frac{\partial L(\mathbf{w})}{\partial w_j}=\frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}+\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})\bullet x_j^{(i)} \qquad (4.10)$$

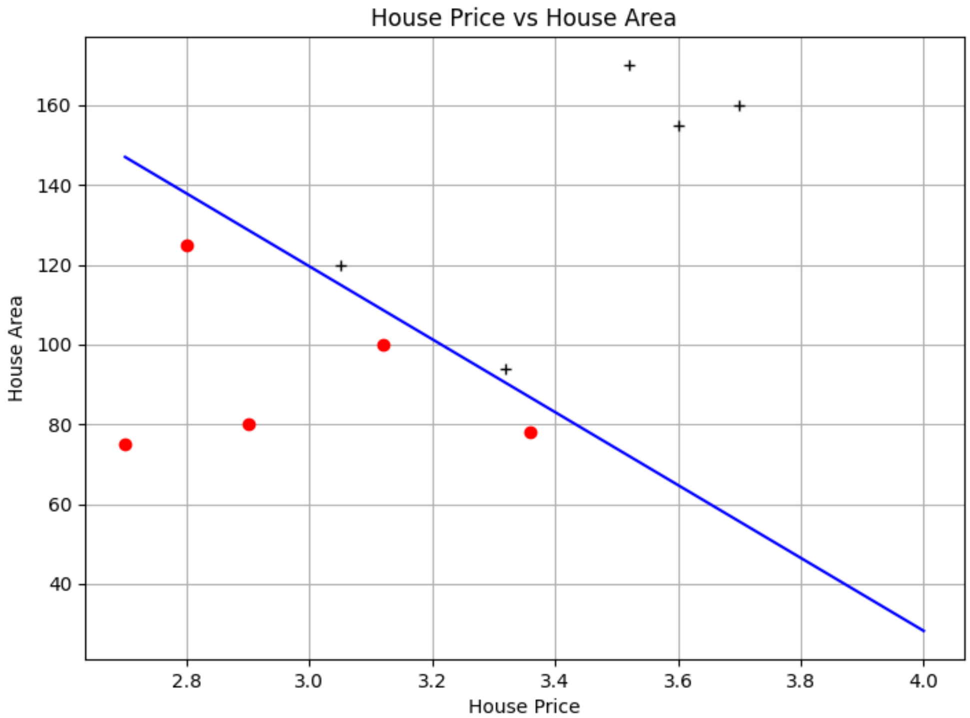
代码4.3

```
#代码 4.3 基于梯度下降法自定义求解房屋好卖预测问题
import numpy as np
import matplotlib.pyplot as plt
from bgd_optimizer import bgd_optimizer

def normalize(X, mean, std): #对数据进行归一化的函数
    return(X-mean)/std

def make_ext(x): #对 x 进行扩展，加入一个全 1 的列
    ones = np.ones(1)[:, np.newaxis] #生成全 1 的向量
    new_x = np.insert(x, 0, ones, axis = 1)
    return new_x

def logistic_fun(z):
    return 1./(1 + np.exp(-z))

def cost_fun(w, X, y):
    tmp = logistic_fun(X.dot(w)) #线性函数，点乘
    cost = -y.dot(np.log(tmp) - (1 - y).dot(np.log(1 - tmp)))
    return cost

def grad_fun(w, X, y): #套用公式 4.12 计算 w 的梯度
    loss = X.T.dot(logistic_fun(X.dot(w)) - y) / len(X)
    return loss
```

```
mean = xTrain.mean(axis = 0) #训练数据平均值
std = xTrain.std(axis = 0) #训练数据方差
xTrain_norm = normalize(xTrain, mean, std) #归一化数据

np.random.seed(0)
init_W = np.random.random(3) #随机初始化 w，包括 w0, w1, w2
xTrain_ext = make_ext(xTrain_norm)

#调用第三章定义的批量梯度下降函数
iter_count, w = bgd_optimizer(cost_fun, grad_fun, init_W, xTrain_ext,
yTrain, lr = 0.001, tolerance = 1e-5, max_iter = 100000000)

w0,w1,w2 = w

#绘制分类分割线，
x1 = np.array([2.7, 3.3, 4.0])
x1_norm = (x1 - mean[0]) / std[0]
x2_norm = -(w0 + w1 * x1_norm) / w2 #拟合曲线：w0+w1*x1+w2*x2=0
x2 = std[1] * x2_norm + mean[1] #由于绘制图时采用原始的 x 值，因此需
要进行“去归一化”
plt.plot(x1, x2, 'b-')
plt.show()
```

Python编码实现

- 演示运行代码4.3：基于梯度下降求解房价预测问题的Python实现



迭代次数： 176589

参数w0, w1, w2的值:

-1.9407385001273494 -3.8817781054764713 -4.408330368012581

输出结果的说明

- 该代码采用批量梯度下降法相同的实现，即bgd_optimizer函数。该函数需要传入成本函数（目标函数）、梯度函数、参数初始值、学习率等通用参数。具体到逻辑斯蒂分类问题，其成本函数和梯度函数已经在前面定义并用Python实现，作为参数传入bgd_optimizer函数即可。

- 由于“房屋面积”与“房屋单价”这两个属性具有不同取值范围，取值范围差异大，在样本数据传入梯度下降函数进行训练之前先进行归一化操作

$$x\_norm_i = \frac{x_i - \overline{x_i}}{std(x_i)}$$

输出结果的说明

- 训练数据的两个属性分别对应优化参数 w_1 和 w_2 ，由于参数向量也包含 w_0，而 w_0 与 1 对应，因此为了便于向量运算，在训练数据属性向量中增加一个值为 1 的量，对应代码中的make_ext()函数。

- 根据输出文字结果，梯度下降法总共迭代了176589次，得到的 w_0 、w_1 、w_2 三个参数值约为-1.94、-3.88、-4.41，得到的逻辑斯蒂分类函数为

$$f(x)=\frac{1}{1+e^{-(-1.94-3.88x_1-4.41x_2)}}$$

回顾整个过程

1. 问题建模
   - 回归、分类 ...

2. 收集数据
   - 回归：特征数据X，连续数据y
   - 分类：特征数据X，离散数据y

3. 特征预处理
   - 归一化
   - 类别特征
   - 时间特征
   - 图像数据、序列数据、图结构数据

4. 构建模型

   模型选择：线性回归、Logistic Regression

   损失函数：均方误差，交叉

5. 模型验证&参数调优

   使用训练数据训练模型，使用验证数据进行参数调优

   验证指标：回归（wmape、R2、均方误差）

6. 模型上线/AB测试

   在测试数据上进行模型测试（测试数据和训练数据来自于同一分布）

思考

- 假设有一类特征属于类别特征，比如房屋装修风格（中式、日式、简约、复古)

- 如何将这一类特征量化来输入模型?