南京邮电大学 Nanjing University of Posts and Telecommunications

Python程序设计（混合式）

for

张伯雷

南京邮电大学计算机学院，数据科学与工程系

https://bolei-zhang.github.io/course/python-ai.html

2024/11/22

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

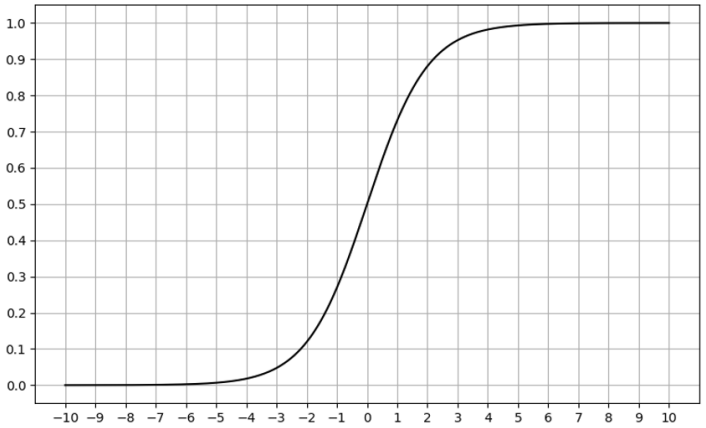
- 让人工智能学习分类是一个复杂的过程，需要**优秀的模型**、**海量的数据**和**高性能的硬件**支持

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 $x=(x_1; x_2; x_3; \ldots; x_d)$

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

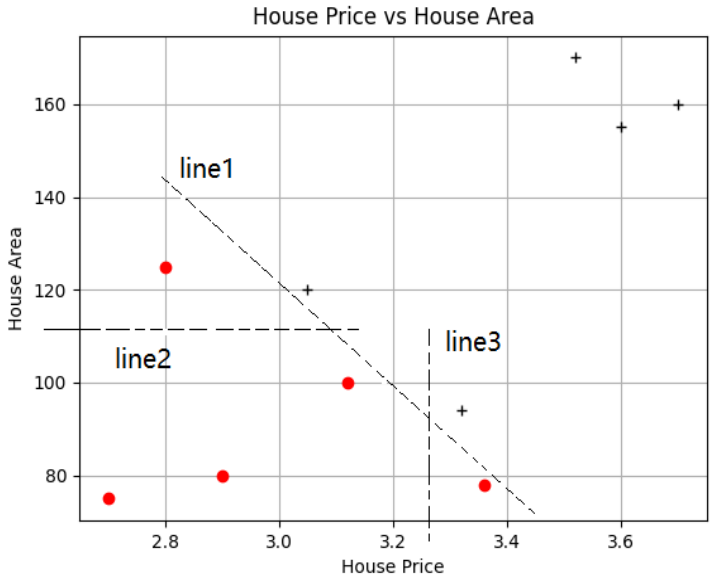
二分类逻辑斯蒂分类问题

- 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]
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],
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()
```

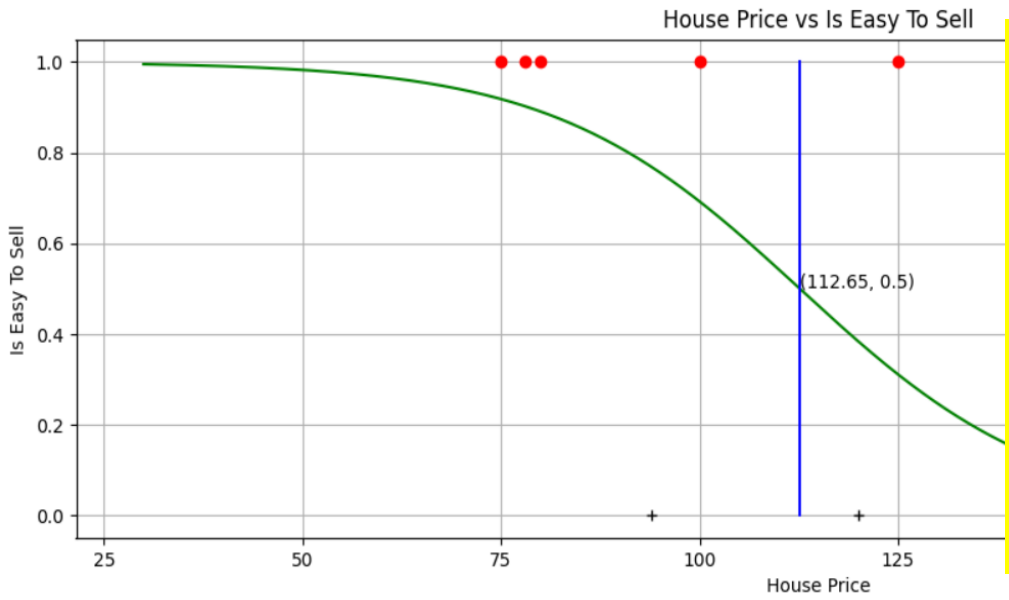
代码4.2

- 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

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

  - ✓ 其中:

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

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

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

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

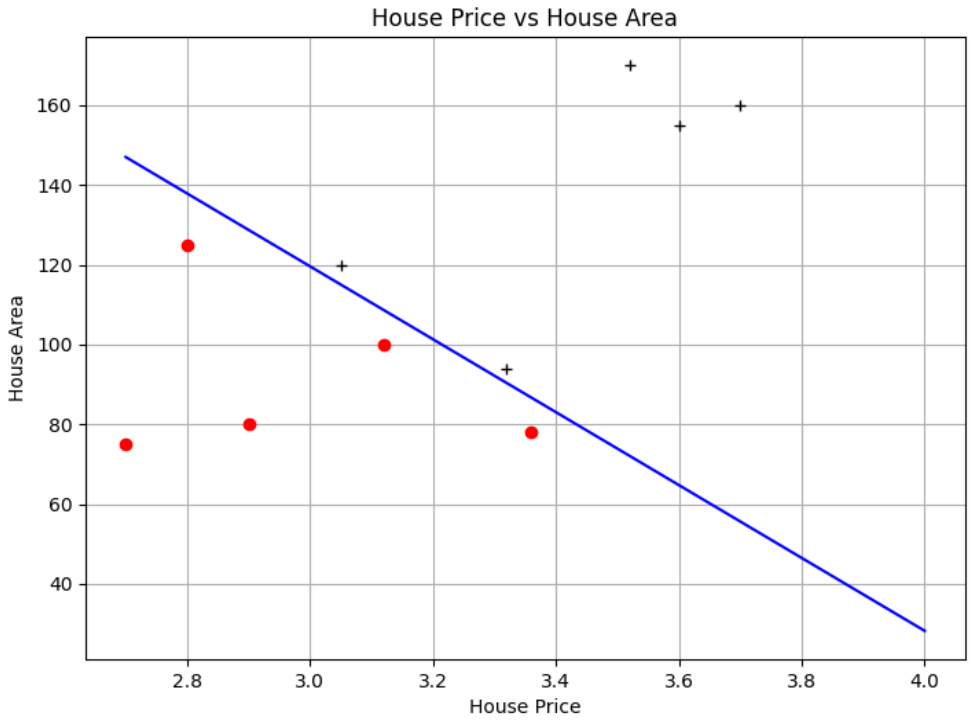
代码4.3

```
#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，交叉熵

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？

目录

分类模型的评价方法

- 对于一些疾病（如癌症、艾滋等），人群中只有约0.5%的人患有这种疾病。因此，完全可以设计一个极端的方法：对于任何样本，永远预测y=0，即该样本没有该疾病。这样的简单方法只有0.5%的错误率，根据这种错误率是否能判定这样的极简模型是好模型呢?

- 对于软件漏洞风险预测，我们希望尽可能多的识别所有的风险，即使一些正常的代码被误判为风险，也可以用人工进行二次校验；但是一些风险如果被错过，则有可能造成非常严重的后果。

- 简单地使用错误率或正确率（Accuracy）来判定模型好坏不一定是一种合适的做法。

二分类模型的评价方法

		真实类别（Actual Class）	
		1	0
预测类别（Predicted Class）	1	真阳（True Positive）	假阳（False Positive）
	0	假阴（False Negative）	真阴（True Negative）

- 正确率（Accuracy）：$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$
- 精准率（Precision）：$\Pr ecision = \frac{TP}{TP + FP}$
- 召回率（Recall）：$\mathrm{Re}call = \frac{TP}{TP + FN}$
- F1指数（F1-Score）：$F1Score = \frac{2PR}{P + R}$

设定测试样本总数为1000个，其中有5个样本为真实确诊病例（$y = 1$），如果根据极端分类方法将所有样本预测为y=0，则TP=0，TN=995，FP=0，FN=5

代码4.4

```
# 代码4.4 手动计算房屋好卖预测的各种评价指标
# 该代码不能独立执行，请粘贴到代码4.3的plt.show()语句之前再运行
xTest = np.array ( [ [ 3.00, 100 ] , [ 3.25, 93 ], [ 3.63, 163 ], [ 2.82, 120 ], [ 3.37, 89 ] ] )
xTest_norm = normalize ( xTest, mean, std )
xTest_ext = make_ext ( xTest_norm )
yTestProbability = logistic_fun ( xTest_ext.dot ( w ) ). # [0.93, 0.62, 0.00, 0.89, 0.40]
yTestPredicted = yTestProbability > 0.5 #预测值为：1 1 0 1 0
yTest = np.array ( [ 1, 0, 1, 1, 1] )
yTest_real_pred = yTestPredicted == yTest #计算预测值与实际值是否相同

accuracy = np.sum ( yTest_real_pred ) / len ( yTest ) #计算正确率
precision = np.sum (yTest_real_pred * yTestPredicted ) / np.sum ( yTestPredicted ) #计算准确率
recall = np.sum ( yTest_real_pred * yTest ) / np.sum ( yTest ) #计算召回率
f1score = 2 * precision * recall / ( precision + recall )

print ( "预测值：", yTestPredicted )
print ( "实际值：", yTest )
print ( "正确率（Accuracy）：", accuracy )
print ( "准确率（Pecision）：", precision )
print ( "召回率（Recall）：", recall )
print ( "F1Score：", f1score )
```

```
预测值： [True  True  False  True  False]
实际值： [1 0 1 1 1]
正确率（Accuracy）： 0.4
准确率（Pecision）： 0.6666666666666666
召回率（Recall）： 0.5
F1Score： 0.5714285714285715
```

运行结果说明

- 测试数据集中共有5个样本，并给出真实y值用于评价；计算预测y值前，先将属性值归一化，再根据分类模型预测y值。

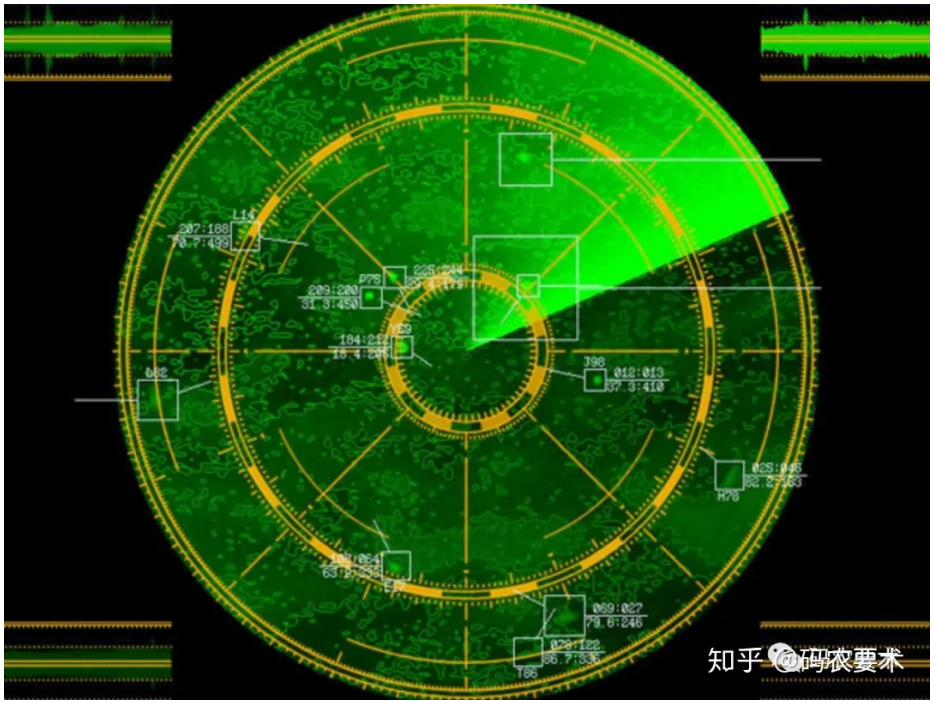
- 语句“yTest_real_pred = yTestPredicted == yTest”是将预测值与真实值逐项比较，相同则对应的项为1，不相同则对应的项为0，注意正例和负例都进行了比较，此时预测值为[1, 1, 0, 1, 0]，真实值为[1, 0, 1, 1, 1]

- **逻辑斯蒂分类模型计算出来的是概率值，判定时需设定一个阈值K，当概率值大于等于K时判定为正例，小于K时判定为负例**

- 思考：请分析，在Covid-19核酸检测筛选问题中，更应看重正确率、准确率和召回率中的哪些指标?

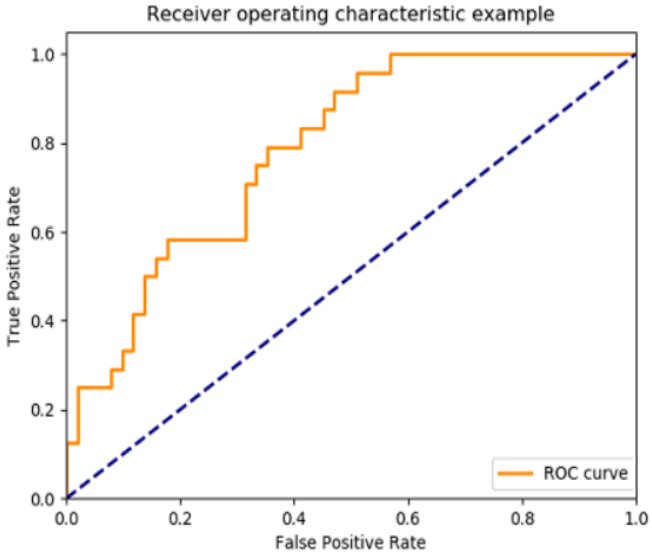
ROC曲线

- ROC最早用于英国雷达分辨鸟或德国飞机的概率。二战期间首次用于分析雷达有效性。在早些时候雷达，有时很难从飞机上分辨出一只鸟。英国人率先使用 ROC 曲线来优化他们依赖雷达进行判别的方式在来袭的德国飞机和鸟类之间的区别



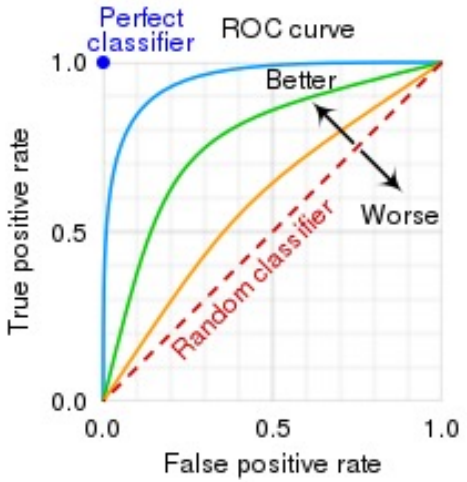
ROC曲线

- 接受者操作特性曲线（receiver operating characteristic curve，简称ROC曲线），又称为感受性曲线（sensitivity curve），起源于二战时期雷达兵对雷达的信号判断



- ROC曲线以**真阳性率（TPR）为纵轴**、**假阳性率（FPR）为横轴**所组成的坐标图，反映模型在选取不同阈值K的时候其真阳性率和假阳性率的趋势走向

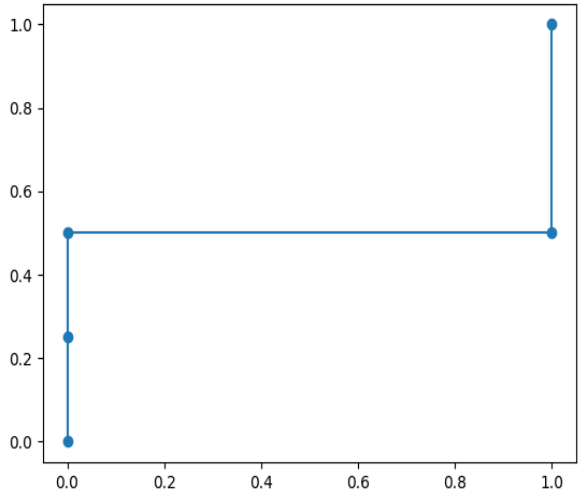
$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$



代码4.5

```
# 代码4.5 绘制房屋好卖预测问题的ROC曲线图
# 该代码需结合代码4.3的模型才能正常运行
xTest = np.array([[3.00, 100], [3.25, 93], [3.63, 163], [2.82, 120], [3.37, 89]])
xTest_norm = normalize(xTest, mean, std)
xTest_ext = make_ext(xTest_norm)
yTestProbability = logistic_fun(xTest_ext.dot(w)) # 0.93,0.62,0.00,0.89,0.40
yTest = np.array([1, 0, 1, 1, 1])
from sklearn import metrics
fpr, tpr, thresholds = metrics.roc_curve(yTest, yTestProbability)
print ( "K值：", ','.join(['%.2f'%(t) for t in thresholds]))
print ( "fpr：", fpr )
print ( "tpr：", tpr )
plt.scatter ( fpr, tpr )
plt.plot ( fpr, tpr )
```

K值： 1.93, 0.93, 0.89, 0.62, 0.00
fpr： [0. 0. 0. 1. 1.]
tpr： [0. 0.25 0.5 0.5 1.]



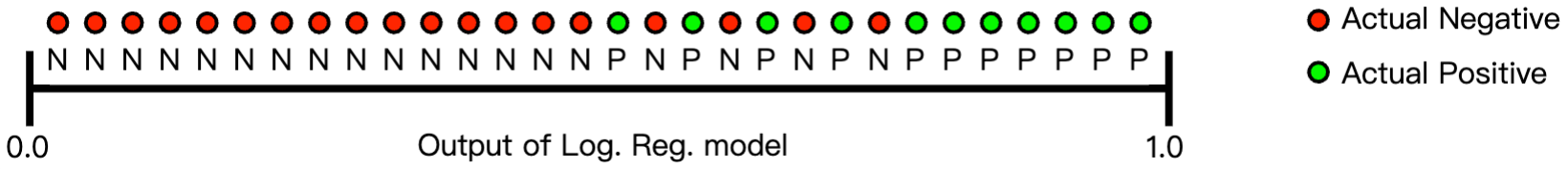
thresholds[0] represents no instances being predicted and is arbitrarily set to max(y_score) + 1

AUC指标

- 理想的情况是：TPR越高越好，FPR越低越好。是否一定递增？

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

- 曲线下的面积被称为AUC （Area under curve）。AUC不同的值分别代表什么？
- AUC固定阈值，允许中间状态的存在；有助于选择最佳的阈值。ROC曲线越靠近左上角，模型的准确性就越高
- AUC取值范围为 0-1
- 物理含义：**AUC表示随机正（绿色）示例位于随机负（红色）示例右侧的概率**[1]

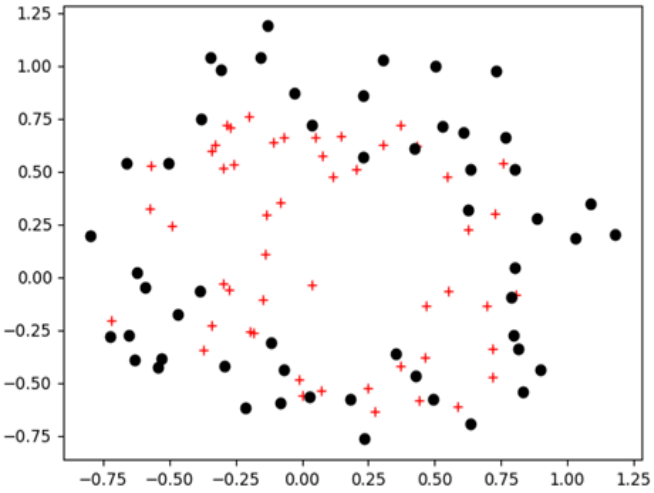


[1] https://kevincodeidea.wordpress.com/2017/01/23/proof-of-the-auc-is-equivalent-to-probability-of-ranking-positives-over-negatives/
[2] https://rogerspy.github.io/2021/07/29/roc-auc/

非线性分类问题

- 在很多情况下，样本数据是线性不可分的，即需要通过一个或多个曲线或曲面才能划分开来

```
#代码4.6 绘制图4.6中的数据图
import numpy as np
import matplotlib.pyplot as plt
#从文本文件加载数据，每行是一个样本，前两列是样本属性，第三列是样本类别，以制表符’\t’隔开，
#数据加载函数loadtxt的delimiter是分隔符，得到的X是多行三列的numpy数组
X = np.loadtxt ( "non-linear-data.txt", delimiter = "\t" )
plt.figure ( )
X11 = X [ X [ :, 2 ] == 1, 0 ] #获取所有类别为1的行的第0个属性值
X12 = X [ X [ :, 2 ] == 1, 1 ] #获取所有类别为1的行的第1个属性值
X01 = X [ X [ :, 2 ] == 0, 0 ] #获取所有类别为0的行的第0个属性值
X02 = X [ X [ :, 2 ] == 0, 1 ] #获取所有类别为0的行的第1个属性值
plt.plot ( X11, X12, "r+" ) #绘制类别为1的样本点
plt.plot ( X01, X02, "ko" ) #绘制类别为0的样本点
plt.show ( )
```



无法找到一条可以将这些正负样本较好分开的直线，这种情况称为“线性不可分”，但从图中可以看出应该可以找到一条曲线将这些样本分隔开

基于Scikit-learn库求解

- 上述数据必须采用高阶曲线才可能进行分割

- 可尝试采用如下逻辑斯蒂分类函数

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

  - ✓ 其中

$$\begin{aligned} g_w(\mathbf{x}) &= w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1x_2 + w_5x_2^2 + w_6x_1^3 + w_7x_1^2x_2 + w_8x_1x_2^2 + w_9x_2^3 + \cdots \\ &+ w_{21}x_1^6 + w_{22}x_1^5x_2^1 + w_{23}x_1^4x_2^2 + w_{24}x_1^3x_2^3 + w_{25}x_1^2x_2^4 + w_{26}x_1x_2^5 + w_{27}x_2^6 \end{aligned}$$

  - ✓ x_1 和 x_2 是样本的两个属性，进行多种乘方组合变化，可以得到共28组不同的特征

代码4.7和代码4.8

```
#代码4.7 采用LogisicRegession类进行分线性分类
import numpy as np
from sklearn.linear_model import LogisticRegression
def extendData ( X0, X1 ):
    rowCount = len ( X0 )
    feature_index = 0
    features = np.zeros ( [ rowCount, 28 ] ) #28=1+2+3+4+5+6+7
    for i in range ( 0, 7 ):
        for j in range ( 0, i + 1 ):
            features [ :, feature_index ] = ( X0 ** ( i – j ) ) * ( X1 ** j )
            feature_index += 1
    return features
data = np.loadtxt ( "non-linear-data.txt", delimiter = "\t" )
newData = extendData ( data [ :, 0 ], data [ :,1 ] )
model = LogisticRegression ( solver = "newton-cg", penalty = "none" )
model.fit ( newData, data [ :, 2 ] )
print ( "model.coef_:  ", model.coef_ )
print ( "model.intercept_:  ", model.intercept_ )
```

```
#代码4.8 绘制代码4.7中模型的分界线
plt.figure()
X11  =data [ data [ :, 2 ] == 1, 0 ]
X12 = data [ data [ :, 2 ] == 1, 1 ]
X01 = data [ data [ :, 2 ] == 0, 0 ]
X02 = data [ data [ :, 2 ] == 0, 1 ]
plt.plot ( X11, X12, "r+" )
plt.plot ( X01, X02, "ko" )
newX1 = np.linspace ( -1, 1.6, 100 )
newX2 = np.linspace ( -1, 1.6, 100 )
Z = np.zeros ( [ len ( newX1 ), len ( newX2 ) ] )
for i in range ( len ( newX1 ) ) :
    for j in range ( len ( newX2 ) ) :
        tmp = extendData ( np.array ( [ newX1 [ i ] ] ),
np.array ( [ newX2 [ j ] ] ) ) #扩展特征
        Z [ i, j ] = model.predict ( tmp ) #计算类别
plt.contour ( newX1, newX2, Z, levels = [ 0 ] )
plt.show ( )
```

“过拟合”问题

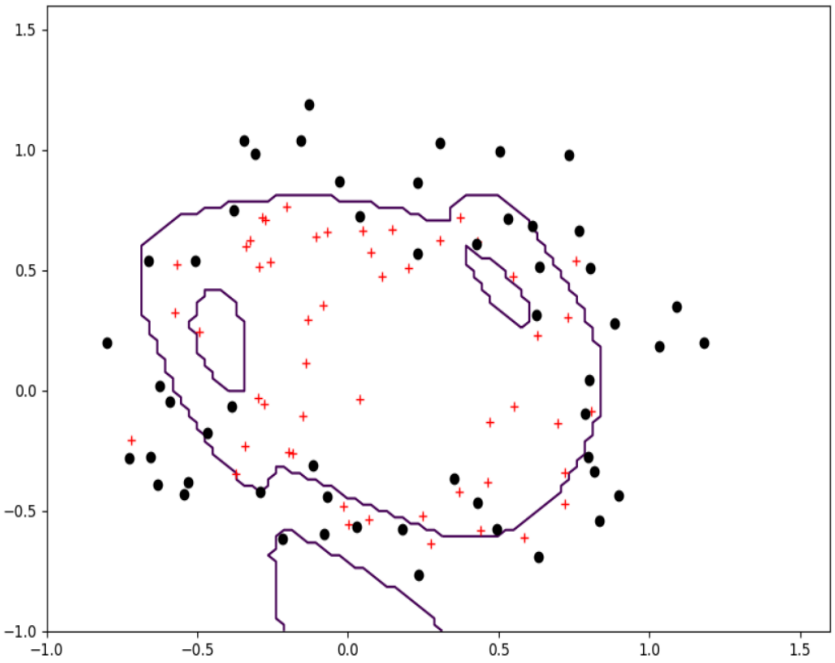
结果

输出：

model.coef_：

[[6.75648906 15.61204317 42.60187485 -61.49758007
-35.68695213 -89.18474107 -13.76242629 -193.87613648
-301.02643472 -244.9038965 96.59149095 132.7132875
425.84753516 254.50022587 307.12501577 -23.98634476
146.08263767 504.59891556 823.70400268 772.78806942
386.27226822 -46.62328301 -74.27162823 -517.93456299
-778.26775291 -1226.28898173 -729.0797253 -444.26963462]]

model.intercept_： [6.75648906]



正则化问题

- **为了防止过拟合问题，正则化方法被提出来了**
- 核心思想是：由于特征数量维度多而造成权重参数也很多，**应尽可能使每个权重参数的值小，以降低模型复杂度和不稳定程度**，从而避免过拟合的危险
- 采用的方法是在成本函数中加入一个惩罚项

$$L(\mathbf{w})=\frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})-(1-y^{(i)})\log(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})]+\frac{\lambda}{2m}\sum_{j=1}^{d}w_j^2$$

- 奥卡姆剃刀定律（Occam's Razor）：如果有多个假设与观察一致，则选最简单的那个
- 过拟合，就是拟合函数需要顾忌每一个点，最终形成的拟合函数波动很大。在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大

正则化项说明

$$L(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}) - (1-y^{(i)})\log(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})] + \frac{\lambda}{2m}\sum_{j=1}^{d} w_j^2$$

- $\frac{\lambda}{2m}\sum_{j=1}^{d} w_j^2$ 是惩罚项，也被称为penalty项，是对参数数量和大小的约束
- 惩罚项是有范式的，以上是二范式项；也可以根据需要替换成零范式（要么有参数 w_j，要么没有参数 w_j）、一范式、三范式
- λ 是超参数，用于调整惩罚项的权重

正则化项说明

$$L(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}) - (1-y^{(i)})\log(1-\frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}})] + \frac{\lambda}{2m}\sum_{j=1}^{d} w_j^2$$

- 惩罚项的梯度是

$$\frac{\partial Penalty}{\partial w_j} = \frac{\lambda}{m} w_j$$

- 得到成本函数的梯度如下

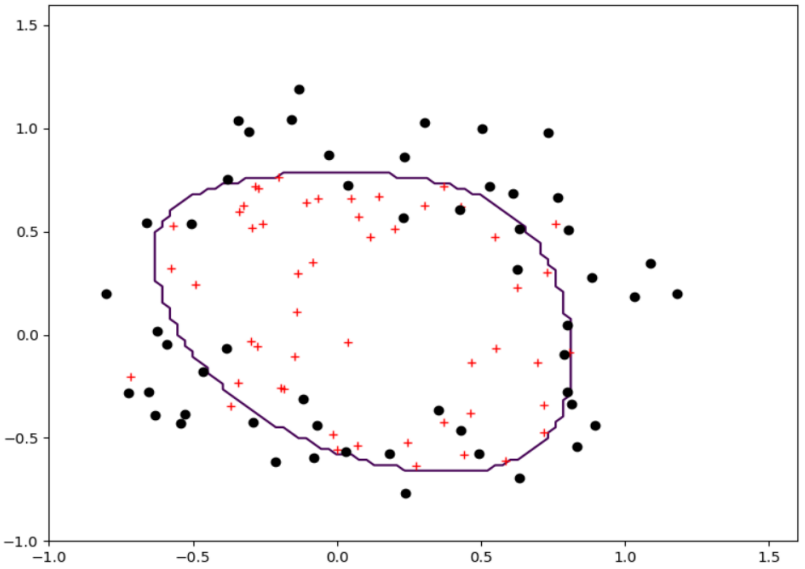
$$\frac{\partial L(\mathbf{w})}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}[(-y^{(i)} + \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}^{(i)}}}) \bullet x_j^{(i)}] + \frac{\lambda}{m} w_j$$

- 该式适用于除 w_0 以外的参数 w_j 的更新

正则化问题的求解实现

- 只需要将其中创建模型对象的语句修改成如下语句即可
  - ✓ model = LogisticRegression (solver = "newton-cg", penalty = “l2") #可选：l1, l3, l4等等

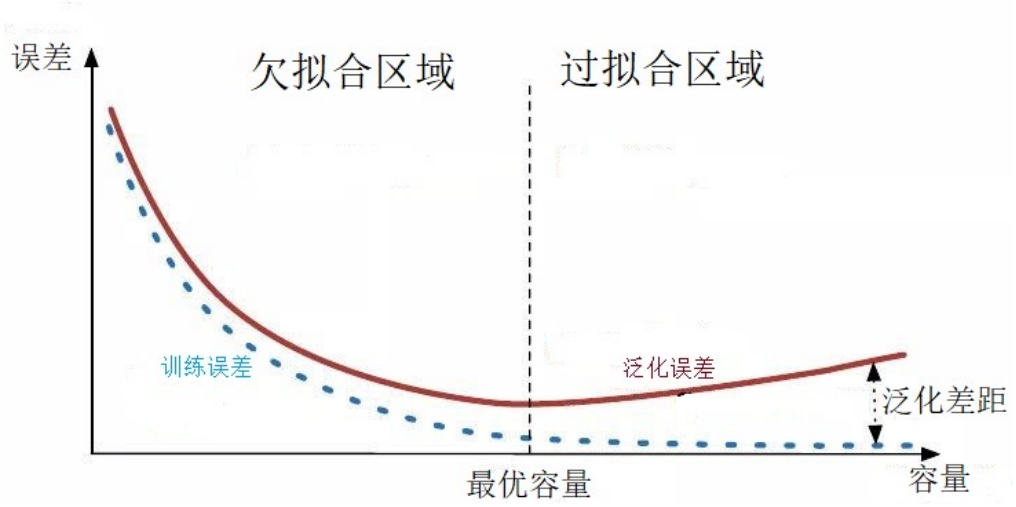
- 运行演示代码4.9



- 由于数据的两个属性值取值范围相同，因此可以省去归一化（normalize）部分；
- 运行后可得到图4.8所示的图，其中分届曲线的过拟合现象有所改善。

防止过拟合的方法

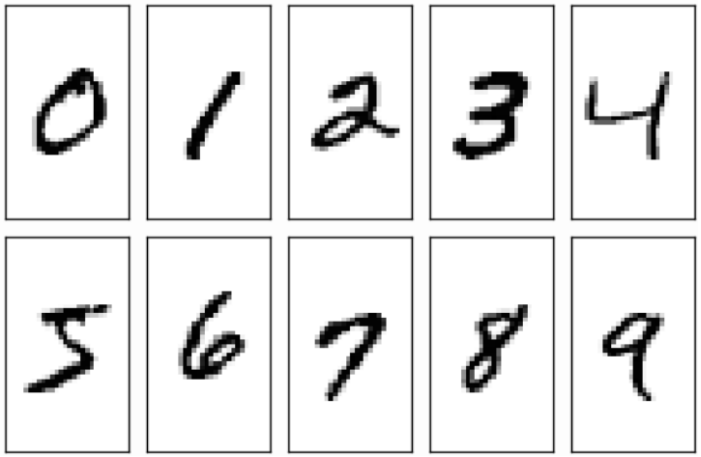
- 过拟合发生的原因：
  - ✓ 训练数据集样本单一，样本不足
  - ✓ 训练数据中噪声干扰过大
  - ✓ 模型过于复杂



- 常用防止过拟合的方法
  - ✓ 使用更多的数据
  - ✓ 控制模型复杂度
  - ✓ 减少特征的数量
  - ✓ 正则化

多类别逻辑斯蒂分类

- 现实真实场景的类别往往是两个以上的
  - ✓ 判断哪种颜色
  - ✓ 判断手写体阿拉伯数字是0~9中的哪一个
  - ✓ 判断手写汉字是哪个汉字
  - ✓ 判断一幅图像中的动物是哪种动物



- 手写阿拉伯数字识别
  - ✓ 本质上是一个有10个类别的多分类问题
  - ✓ 对大量已知图片数据进行训练，得出相应的逻辑斯蒂分类模型参数

求解步骤

- 第一步：准备训练数据和测试数据
  - ✓ 训练数据包含5000张手写数字图片
  - ✓ 测试数据包含500张手写数字图片
  - ✓ 每张图片标注好对应的正确数字，称为标签
  - ✓ 每张图片分辨率统一为28*28
  - ✓ **每张图片都是灰度图，即像素值取值为0~255，0表示纯白，255表示纯黑，中间值为黑度不同的灰色，省去了处理RGB彩色图片的负担**
  - ✓ 数据在arab_digits_for_training.txt和arab_digits_for_testing.txt文件

求解步骤

- 第二步：计算特征矩阵用于模型的训练
  - ✓ 将每个像素当做一个特征，每张图对应一维数组，将图片的每个像素点值依次连续放在数组中，共得到28*28个数组元素；图片的标签放在数组最后；**最终一维数组的大小是28*28+1=785**
  - ✓ 5000个训练图片对应一个二维数组，大小是5000*785，构成一个训练矩阵
  - ✓ 因为像素值取值范围是0~255，因此对训练数据进行归一化处理（normalization）

- 第三步：训练逻辑斯蒂分类模型并评估

基于Scikit-learn库实现

```
#代码4.10 使用LogisicRegession实现多类别分类
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
def normalize ( X, col_means ):
    return ( X - col_means ) / 255
#加载训练数据
trains = np.loadtxt ( "arab_digits_training.txt", delimiter = "\t" )
trainX = trains [ :, 1: ] #第0列是标签
trainY = trains [ :, 0 ]
col_means = np.mean ( trainX, 0 ) #每个属性的平均值
trainX = normalize ( trainX, col_means )
model = LogisticRegression ( solver = "lbfgs", multi_class = "multinomial", max_iter = 500 )
model.fit ( trainX, trainY )
```

```
#加载测试数据
tests = np.loadtxt ( "arab_digits_testing.txt", delimiter = "\t" )
testX = tests[ :, 1: ] #第0列是标签
testY = tests [ :, 0 ]
testX = normalize ( testX, col_means )
predictY = model.predict ( testX )
errors = np.count_nonzero ( testY – predictY )
print("预测错误数是：{}/{}".format ( errors, np.shape ( tests ) [ 0 ] ) )
```

预测错误数是：54/500

- 代码中的归一化函数normalize( )，其参数是一个二维numpy数组，对其中的每个列（属性）都要进行归一化；其目的是为了防止计算结果经过层层乘法运算后得到的数值过大而导致成本函数溢出
- 原则上采用前面章节的归一化公式

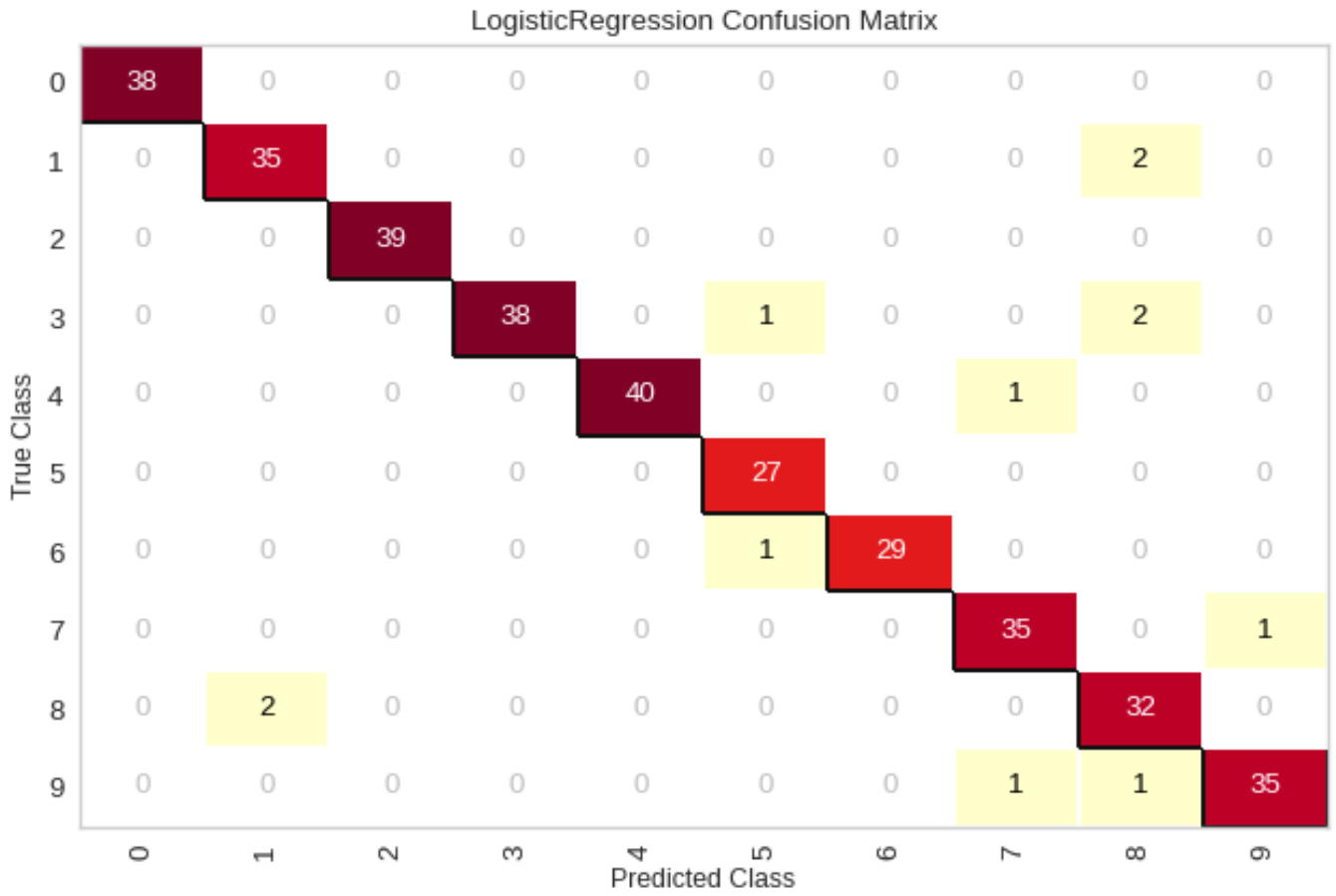
基于梯度下降法求解

- 由于原始的逻辑斯蒂回归解决的是二分类的问题，为了解决手写数字识别的多分类问题，可以把多分类问题转化为二分类问题求解
  - ✓ 为每个类别计算一个双类别逻辑斯蒂分类模型，其中该类别为正，其它所有类别为负，得到10个具有不同参数的双类别逻辑斯蒂模型
  - ✓ 在预测时，使用这10个模型对每个类别计算得到不同的预测概率，取其中概率最大的类别为预测类别

- 演示代码4.11
  - ✓ 预测错误数是：53/500

思考

- 多分类问题用什么指标进行评估



总结

- 逻辑斯蒂分类（二分类、多分类）定义
- 求解逻辑斯蒂分类的多种方法
  - ✓ Scikit-Learn库函数求解法
  - ✓ 梯度下降法
- 分类问题的评价指标
  - ✓ 正确率
  - ✓ 准确率
  - ✓ 召回率
  - ✓ ROC曲线

- 作业：Spoc第四章