**Spark机器学习回归算法基础**

**教案**

|  |  |  |
| --- | --- | --- |
| **课程教案版本** | **日期** | **备注** |
| **V1.0** | **20190515** |  |
|  |  |  |

# SparkMliib实战回归算法

学习目标：

1.SparkMllib的Pipeline原理及实践

2.SparkMllib模型选择与优化

3.线性回归原理及实战

4.逻辑回归原理及实战

## 2.SparkMliib构建回归算法

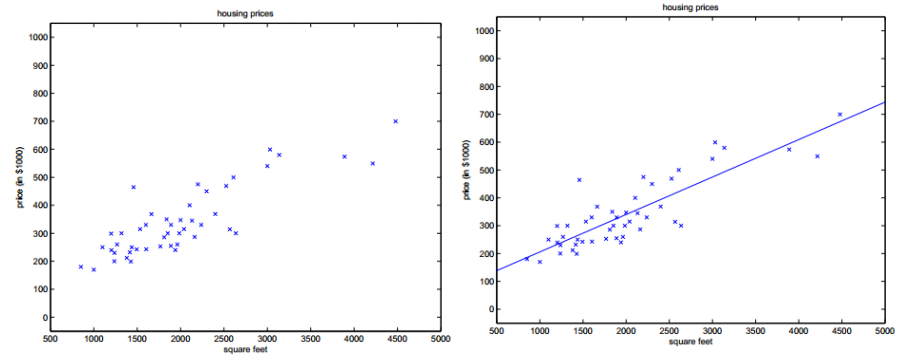
### 2.1 SparkMllib回归原理详解

【回忆】监督学习分为几类问题？

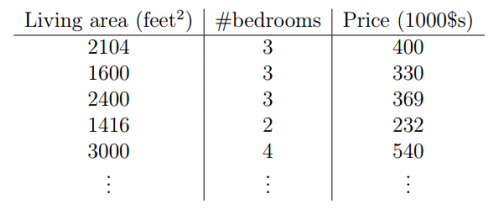
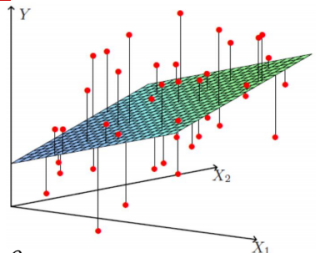
#### 2.1.1SparkMllib线性回归原理及实战

##### （1）房价问题

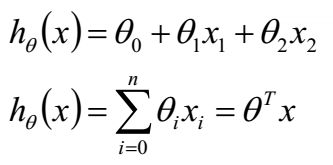
考虑房屋的面积和房屋的售价，这里我们可以用y=kx+b预测



模拟数据如下：

建立房屋面积、卧室数目和售价的模型（思考：如何使得模型最优？）：



当有了新的房屋面积数据即可预测房价。

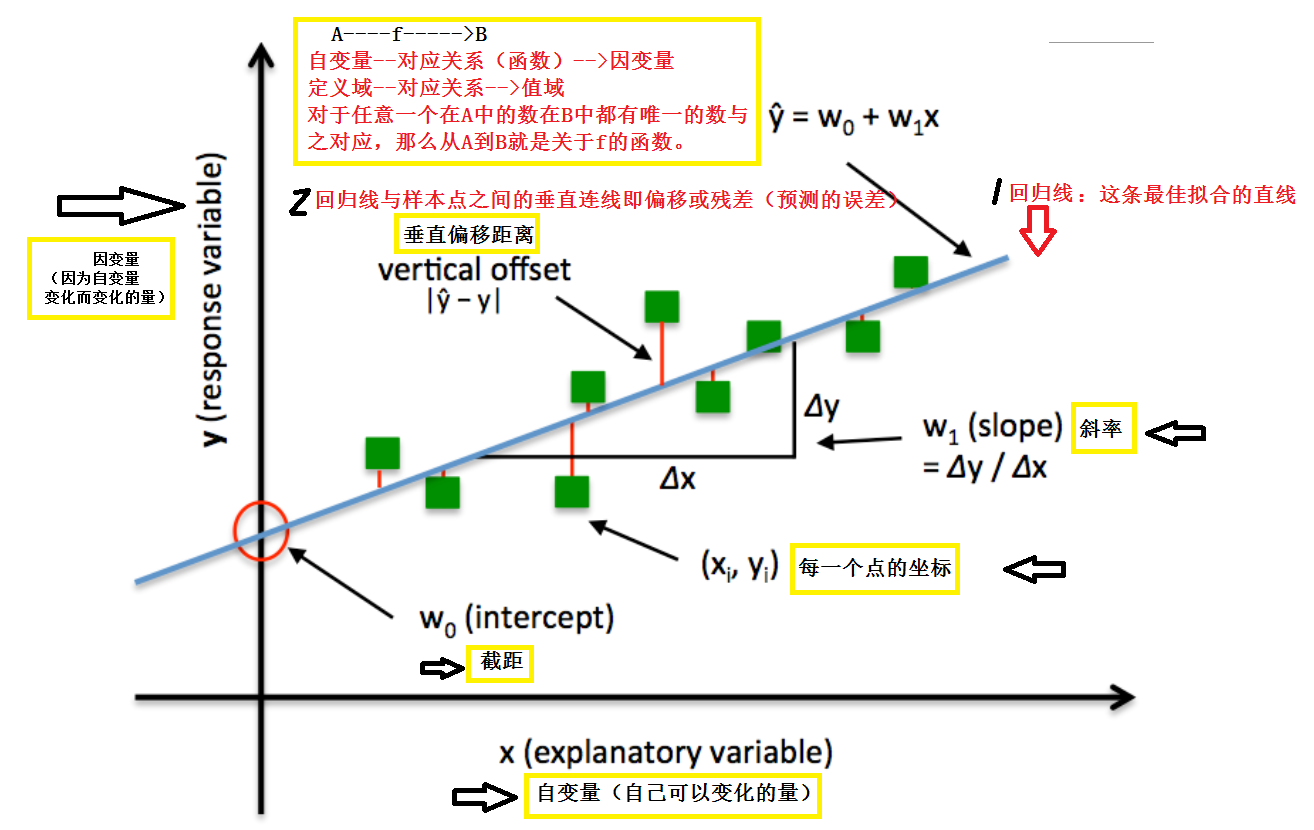
##### （2）回归模型

回归模型（regression model）用于连续型变量的预测分析。

通常在只有一个解析变量的时候，线性回归称为简单线性回归，如y=kx+b；当有多个解析变量的时候，称之为多元线性回归，，其中w0是x0=1时在y轴上的截距。



**线性回归模型的目标：**通过学习得到线性方程的这两个权值，如在y=kx+b中，得到k和b两个权值，并用这个方程解释变量和目标变量之间的关系。



##### （3）简单线性回归

我们都参加过高考，据统计，高考的物理成绩确实与数学成绩有一定关系，但除此之外，还存在很多影响物理成绩的因素，例如：是否喜欢物理，用在物理上的时间等。而当我们主要考虑数学成绩对物理的影响时，就是要考察这两者之间的相关关系。

现实生活中还有很多的相关关系，如

1.商品销售输入与广告支出经费之间的关系，销售输入与广告支出有着密切的关系，但是还与商品质量、居民收入等因素有关。

2.粮食产量与施肥量之间的关系。在一定范围内，施肥量越大，粮食生产就越高。除此之外，粮食产量还受到土壤质量、降雨量等的影响。

3.人体内脂肪的含量与年龄之间的关系。在一定年龄段内，随着年龄的增长，人体内的脂肪含量会增加，但人体内的脂肪含量还和饮食习惯，体育锻炼有关系，可能还与先天

体质有关系。

对于上述两个变量之间的关系，应该说都可以根据经验做出相应的判断，因为“经验当中有规律”，但是，不管你经验多么丰富，如果只凭借经验办事，还是很容易出错。因此在分析两个变量之间的相关关系时，我们需要一些说服力的办法。

在寻找变量之间的相关关系中，统计同样发挥着非常重要的作用。因为上面提到的这种关系，并不像匀速直线运动中时间与速度的关系那样是完全确定的，而是带有不确定性，这就需要通过收集大量的数据（有时候通过调查、或实验），在对数据进行统计分析的基础上，发现其中的规律，才能让他们之间的关系做出判断。

**两个变量的线性相关：**

如下表中描述了人体的脂肪百分比和年龄的关系图表：

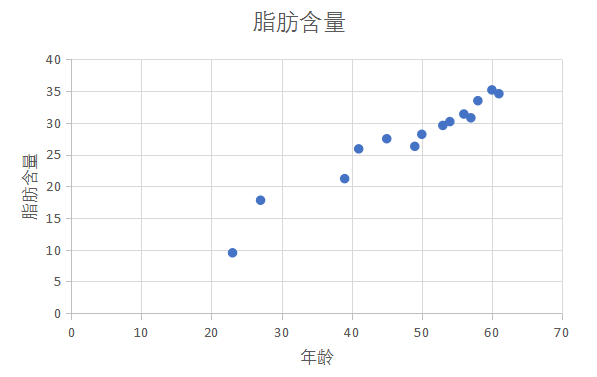
|  |  |
| --- | --- |
| 年龄 | 脂肪含量 |
| 23 | 9.5 |
| 27 | 17.8 |
| 39 | 21.2 |
| 41 | 25.9 |
| 45 | 27.5 |
| 49 | 26.3 |
| 50 | 28.2 |
| 53 | 29.6 |
| 54 | 30.2 |
| 56 | 31.4 |
| 57 | 30.8 |
| 58 | 33.5 |
| 60 | 35.2 |
| 61 | 34.6 |

**问题：**根据上述数据，人体的脂肪含量与年龄之间有怎么样的关系呢？

**对问题的描述：**

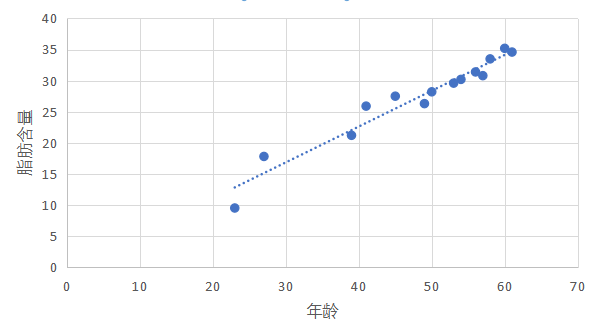
一般地，对于某个人来说，他的体内脂肪不一定随年龄增长而增加或减少，但是如果把很多个体放在一起，这时就可能表现出一定的规律性，各年龄对应的脂肪数据是这个年龄人群脂肪含量的样本平均值。观察上述表数据，从大体上看，随着年龄的增加，人体中脂肪的百分比也在增加。为了确定这一细节，我们需要进行数据的分析，与以前一样，我们可以做统计图、表，通过作统计图、表，可以使我们对两个变量之间的关系有一个直观的印象和判断。

下面我们做一个散点图，如图，假设人的年龄影响体内脂肪含量，于是，按照习惯，以x轴表示年龄，y轴表示脂肪含量，得到下图：



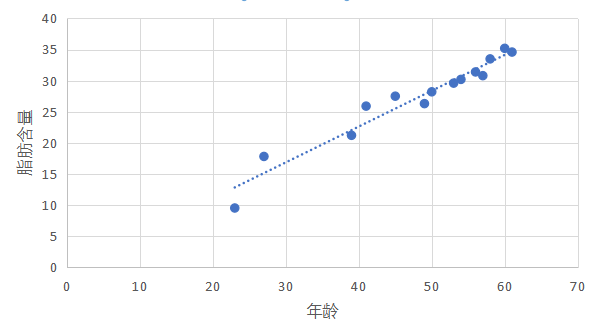
这些散分布的位置也是值得注意的，他们散布在从左上角到右下角的区域。对于两个变量的这种相关关系，我们称为正相关。还有一些变量，例如汽车的重量和汽车每消耗1L汽油所行驶的平均路程，是负相关，也就是汽车越重，每消耗1L汽油所行驶的平均路程就越短，这时的点如果绘制在画布上 将会从左上角到右下角的区域内。

接下来，需要进一步考虑的问题是，当人的年龄增加时，体内脂肪含量到底是以什么方式增加的呢？



从散点图可以看出，这些点大致分布在通过散点图中心的一条直线附近，如果散点图中的点分布从整体上看大致是在一条直线附近，我们就称这两个变量之间具有线性相关关系，这条直线叫做回归直线（regression line）。

如果能求出这条回归直线的方程，那么我们就可以清晰的了解年龄与体内脂肪含量的相关性，就像平均值可以作为一个变量的数据的代表一样，这条直线可以作为两个变量具有线性相关性的代表。



当你拿到这样一个任务的时候，你可能会采用测量的做法，先画出一条直线，测量各点与它的距离，然后移动直线，到达一个使得距离的和最小的位置，测量出此时的斜率和截距，既可以得到回归方程了。

也可能会采用平均方法，也就是在散点图中多取几组点，确定出几条直线的方程，在分别求出各条直线的斜率、截距的平均数，将这两个平均数作为回归方程的斜率和截距。

上面方法虽然有一定道理，但是总让人感觉可靠性不强。实际上，求回归方程的关键是如何用数学的方法来刻画“从整体上，各点与此直线的距离最小”。

假设我们已经得到两个具有线性相关关系的变量的一组数据



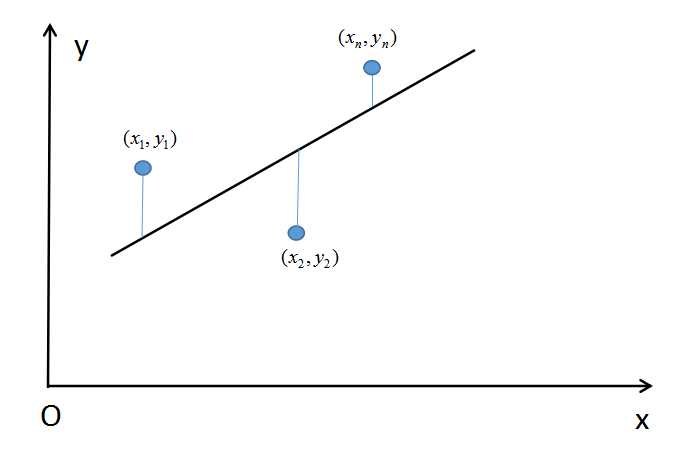
且所求的回归方程设为



它与实际数据的偏差是



在坐标轴中表示如下



这样，用这n个偏差的和来刻画“各点与此直线的整体偏差”比较合适的，由于可正可负，为了避免互相抵消，可以考虑用代替，但由于它含有绝对这，运算不方便，所以改用

即：



这样，问题就归结为：当a和b取什么值时Q最小，即总体偏差最小，经过数学上的最小值运算，a和b的值由下面式子给出



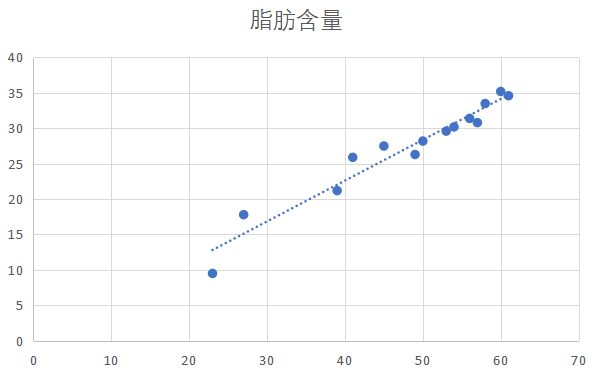


其中b是回归方程的斜率，a是截距。

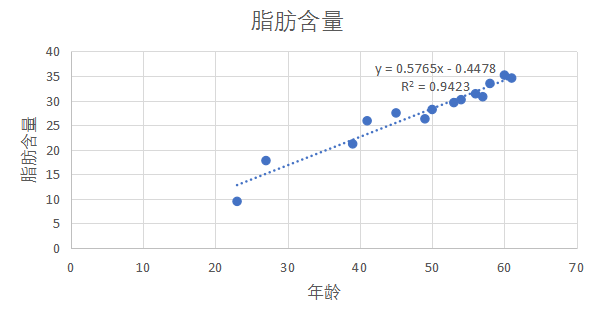
***总结：***这种通过求解Q关系式的最小值而得到回归直线的方法，即求**回归直线**，使得样本数据的点到它的距离的平方和最小的方法，叫做最小二乘法（Least square）。

***实操***：演示Execl画回归拟合直线：

1. 准备数据，首先绘制散点图
2. 在散点图基础上点”趋势预测”即可看到拟合的直线。



1. 在图示区域显示拟合直线的方程



##### （4）简单线性回归模型表示

  （1）被用来描述因变量(y)和自变量(X)以及偏差(error)之间关系的方程叫做回归模型

（2） 简单线性回归的模型是:



  其中：参数，为偏差

**注意：关于偏差ε的假定**

     （1）是一个随机的变量，均值为0，E（ε）=0

     （2）ε的方差(variance)对于所有的自变量x是一样的

     （3）ε的值是独立的

     （4）ε满足正态分布

如果求解简单线性回归方程，如下：

    数学方程：E(y) = β0+β1x

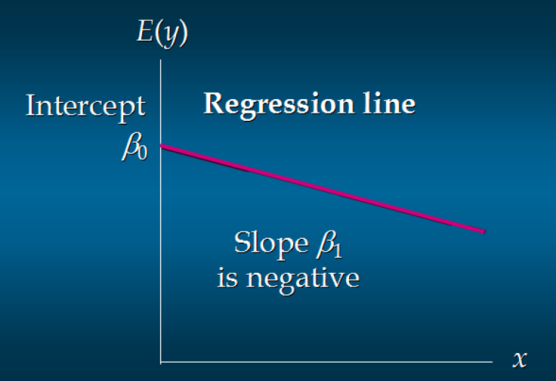
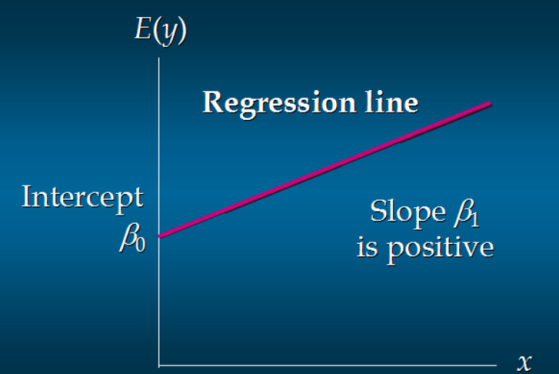
    这个方程对应的图像是一条直线，称作回归线

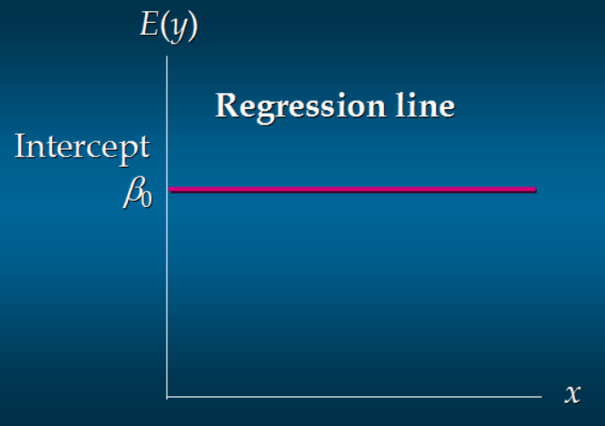
    其中，β0是回归线的截距

        β1是回归线的斜率

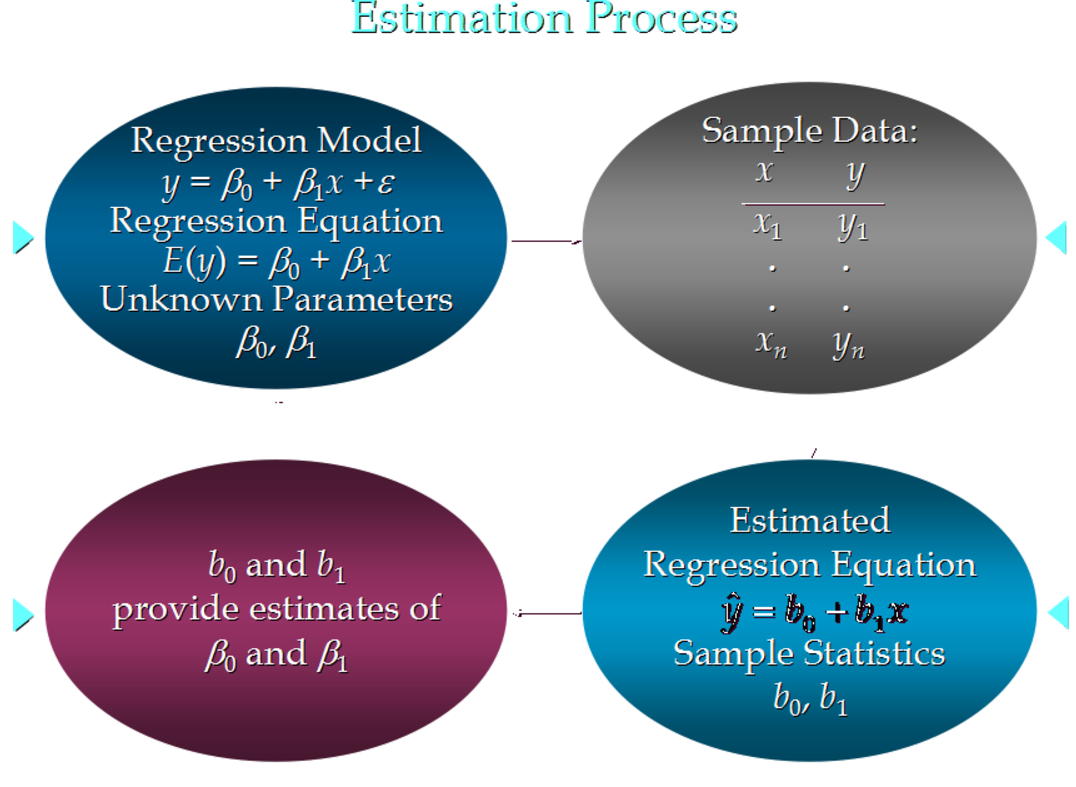
        E(y)是在一个给定x值下y的期望值（均值）

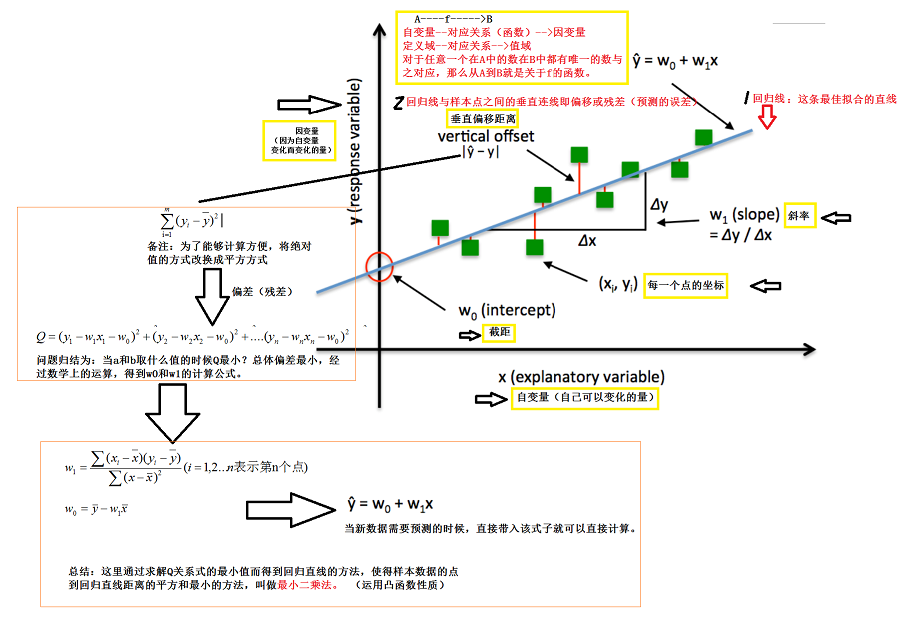
**几种线性表示图解：**





**通过下图了解下线性回归分析流程：**





**接下来完成练习题熟悉简单线性回归内容：**

如下统计表为卖出的热饮杯数与当天气温的对比表：

根据图表回答以下问题：

画出散点图。

从散点图中发现气温与卖出热饮杯数的一般规律。

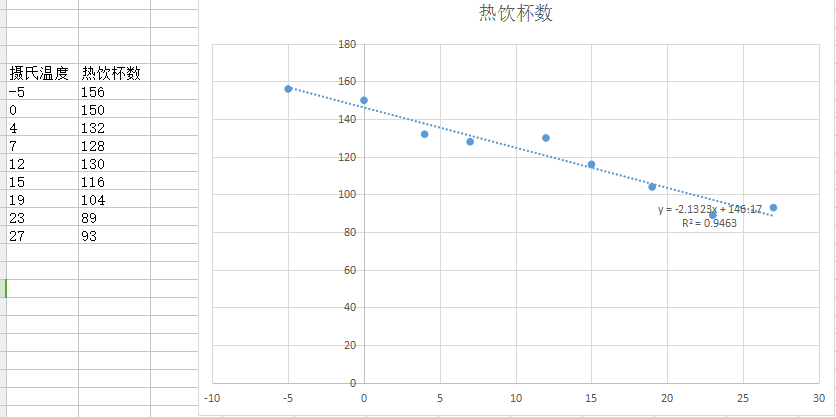
求回归方程。

如果某天气温是2度，预测这天卖出的热饮数。

|  |  |
| --- | --- |
| 摄氏温度 | 热饮杯数 |
| -5 | 156 |
| 0 | 150 |
| 4 | 132 |
| 7 | 128 |
| 12 | 130 |
| 15 | 116 |
| 19 | 104 |
| 23 | 89 |
| 27 | 93 |
| 31 | 76 |
| 36 | 54 |

解答过程：

（1）拟合直线：



1. 从上图中可以看出各点散布在从左上角到右下角的区域，因此气温与热饮销量呈负相关，即气温越高，销售的热饮数量越少。
2. 看图中的直线方程
3. 当x=2时，，因此，某天的气温为2摄氏度，这天大概可以卖出143杯热饮。

##### （5）简单线性回归模型举例及代码

**1.****汽车销售和投放广告销售预测**

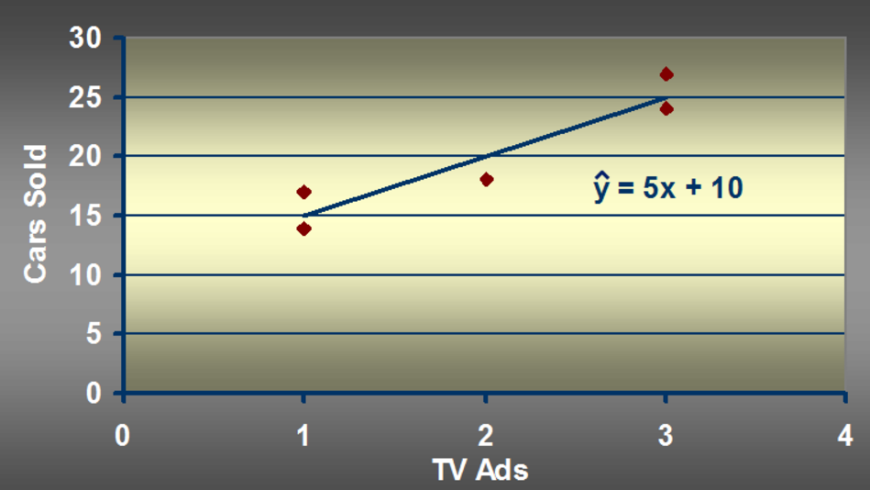
（1）源数据

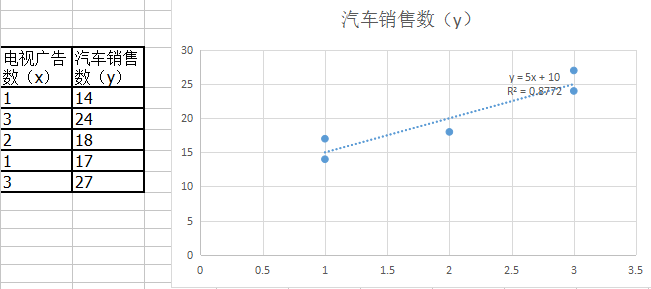
汽车卖家做电视广告数量与卖出的汽车数量：

|  |  |
| --- | --- |
| 电视广告数（x） | 汽车销售数（y） |
| 1 | 14 |
| 3 | 24 |
| 2 | 18 |
| 1 | 17 |
| 3 | 27 |
| Sum（x）=10，Avg（x）=2 | Sum（y）=100，Avg（y）=20 |

（2）找出能够拟合上述数据的最佳直线

如何找出适合简单线性回归模型的最佳回归线？





这里我们通过构造以下函数并使得该函数最小来求解最佳拟合的直线



如果使得上述函数最小，这里我们借助我们高中学习的导数方法，先求导并令导数为0来求解最佳的拟合直线的参数，如一次项的系数和常数项的值。

这里避免过于复杂的数学推导，直接给出一次项系数的参数和偏差项

，

1. **计算**

|  |  |
| --- | --- |
| 电视广告数（x） | 汽车销售数（y） |
| 1 | 14 |
| 3 | 24 |
| 2 | 18 |
| 1 | 17 |
| 3 | 27 |
| Sum（x）=10，Avg（x）=2 | Sum（y）=100，Avg（y）=20 |

分子 = (1-2)(14-20)+(3-2)(24-20)+(2-2)(18-20)+(1-2)(17-20)+(3-2)(27-20)

      = 6 + 4 + 0 + 3 + 7

      = 20

分母 = （1-2）^2 + (3-2)^2 + (2-2)^2 + (1-2)^2 + (3-2)^2

       = 1 + 1 + 0 + 1 + 1

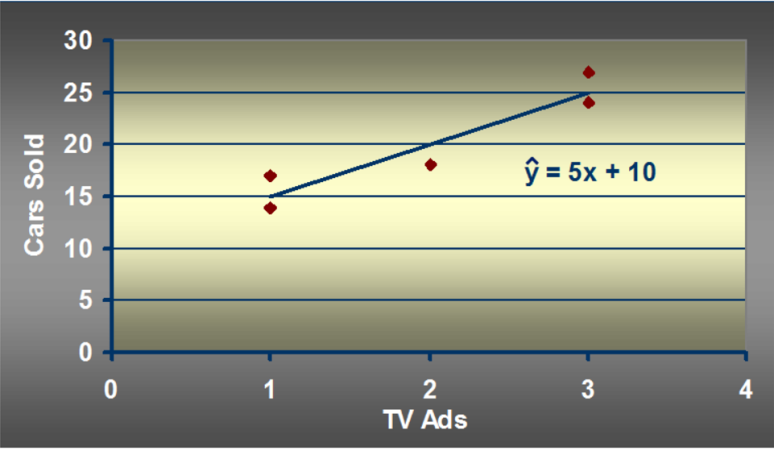
       4

b1 = 20/4  =5

b0=20-5\*2=10

**（4）预测**

假设有一周广告数量为6，预测的汽车销售量是多少？



x\_given = 6

Y\_hat = 5\*6 + 10 = 40

##### （6）简单线性回归代码实现

import numpy as np

def fitSLR(x, y):

    n = len(x)

    dinominator = 0

    numerator = 0

    for i in range(0, n):

        numerator += (x[i] - np.mean(x))\*(y[i] - np.mean(y))

        dinominator += (x[i] - np.mean(x))\*\*2

    b1 = numerator/float(dinominator)

    b0 = np.mean(y)/float(np.mean(x))

    return b0, b1

def predict(x, b0, b1):

    return b0 + x\*b1

x = [1, 3, 2, 1, 3]

y = [14, 24, 18, 17, 27]

b0, b1 = fitSLR(x, y)

print "intercept:", b0, " slope:", b1

x\_test = 6

y\_test = predict(6, b0, b1)

print "y\_test:", y\_test

##### （5）多元线性回归

1.与简单线性回归区别(simple linear regression)：多个自变量(x)

2.多元回归模型表示如下：



3.多元回归方程



4.估计多元回归方程



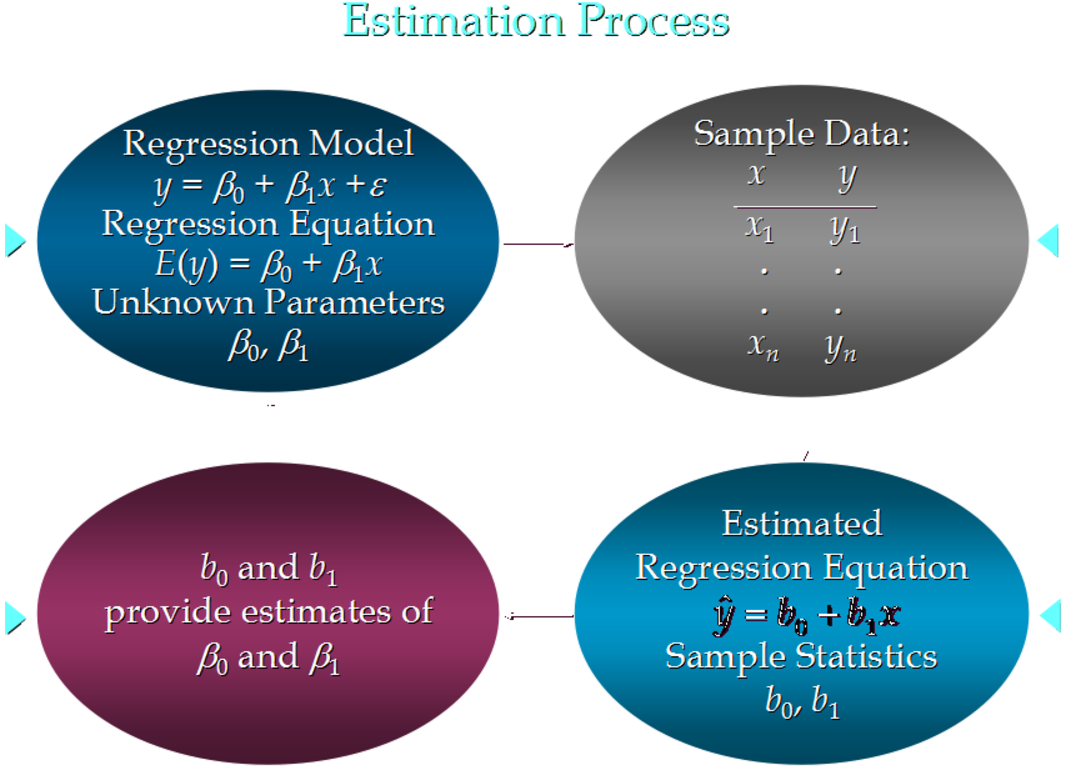
5. 估计方法---误差平方和最小

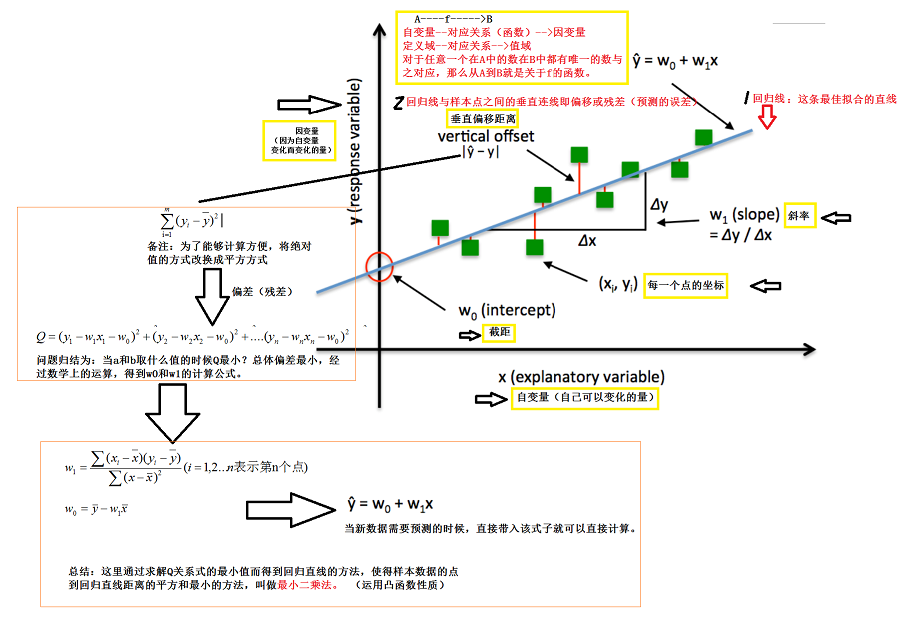
    使sum of squares（平方和）最小



    运算与简单线性回归类似，涉及到线性代数和矩阵代数的运算

6. 估计流程  (与简单线性回归类似）





1. **例子**

7.1问题描述：一家快递公司送货：X1： 运输里程 X2： 运输次数   Y：总运输时间：

|  |  |  |  |
| --- | --- | --- | --- |
| **Driving**  **Assignment** | **X1=Miles**  **Traveled** | **X2=Number of Deliveries** | **Y= Travel Time (Hours)** |
| 1 | 100 | 4 | 9.3 |
| 2 | 50 | 3 | 4.8 |
| 3 | 100 | 4 | 8.9 |
| 4 | 100 | 2 | 6.5 |
| 5 | 50 | 2 | 4.2 |
| 6 | 80 | 2 | 6.2 |
| 7 | 75 | 3 | 7.4 |
| 8 | 65 | 4 | 6.0 |
| 9 | 90 | 3 | 7.6 |
| 10 | 90 | 2 | 6.1 |

Y(Time) = b0+ b1\*Miles + b2 \* Deliveries

Time = -0.869 + 0.0611 Miles + 0.923 Deliveries

7.2. 描述参数含义

     B1: 平均每多运送一英里，运输时间延长0.0611 小时

     B2: 平均每多一次运输，运输时间延长 0.923 小时

7.3. 预测

     如果一个运输任务是跑102英里，运输6次，预计多少小时？

     Time = -0.869 +0.0611 \*102+ 0.923 \* 6= 10.9 (小时）

7.4.处理分类型变量

如果自变量中有分类型变量(categorical data) , 如何处理？

|  |  |  |  |
| --- | --- | --- | --- |
| 英里数 | 次数 | 车型 | 时间 |
| 100 | 4 | 1 | 9.3 |
| 50 | 3 | 0 | 4.8 |
| 100 | 4 | 1 | 8.9 |
| 100 | 2 | 2 | 6.5 |
| 50 | 2 | 2 | 4.2 |
| 80 | 2 | 1 | 6.2 |
| 75 | 3 | 1 | 7.4 |
| 65 | 4 | 0 | 6 |
| 90 | 3 | 0 | 7.6 |

7.5.关于误差的分布

误差ε是一个随机变量，均值为0

ε的方差对于所有的自变量来说相等

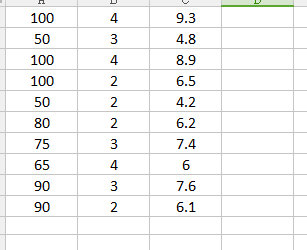
所有ε的值是独立的

ε满足正态分布，并且通过β0＋β１x1+β2x2+ ... +βpxp反映y的期望值

7.6.Python代码实战

数据：

1. 在execl中输入数据另存为csv文件
2. 直接在txt中输入数据以逗号隔开



Python代码：

#-\*-coding:utf8-\*-  
import numpy as np  
from numpy import genfromtxt  
from sklearn import datasets,linear\_model  
#组织数据，通过逗号分隔的CSV文件组织  
datapath=r"D:\delivery.csv"  
delivesyData=genfromtxt(datapath,delimiter=',')  
print "data",delivesyData  
#区分特征和类标签  
X=delivesyData[:,:-1]  
Y=delivesyData[:,-1]  
#打印切分的数据  
print "X",X  
print "Y",Y  
#模型初始化  
regr=linear\_model.LinearRegression()  
regr.fit(X,Y)  
#打印模型的属性  
print "coefficient",regr.coef\_  
print "intercept",regr.intercept\_  
#适用模型进行预测  
xPred=[102,6]  
yPred=regr.predict(xPred)  
print "Y-predicted",yPred

##### （6）回归问题性能评估详解：

不同于类别预测，不能苛刻回归预测的数值结果要严格的和真实值一致。一般情况下，我们希望衡量预测值和真实值之间的差距。因此，可以通过多种测评函数进行评价。

通常最为直观的评价指标包括：

表示回归模型的预测结果，表示样本的真实结果值。

平均绝对误差：





均方误差：





R-Squared：

既考量了回归值与真实值的差异，同时也兼顾了问题的真实值的变动，f(x)代表了回归模型x的预测值。





代表测试数据真实值的方差（内部差异）；代表了回归值与真实值之间的平差差异(回归差异)。因此在统计含以上，R-Squared用来衡量模型回归结果的波动可被真实值验证的百分比，也暗示了模型在数值回归方面的能力。

##### （7）极大似然估计和高斯分布推导最小二乘法

似然函数：

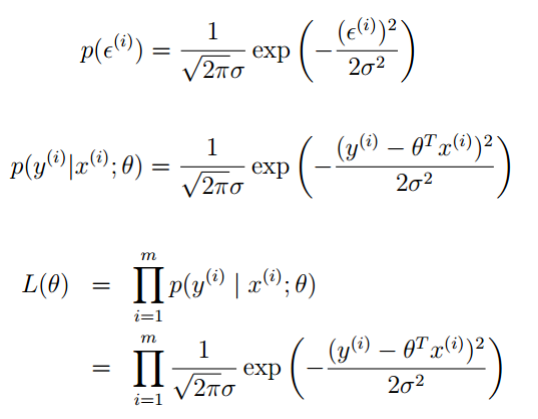
第i个样本的x和第i个样本的预测值y，是误差，表示的是第i个样本的真实值和预测值之间的差距，比如房子的预测价格和实际价格。

在实际应用中都是独立同分布的，**服从**均值是0，方差是sima\*\*2的高斯分布（正态分布）。利用房屋售价理解，所有房屋价格的平均值就是目前房屋的正常的价格范围。

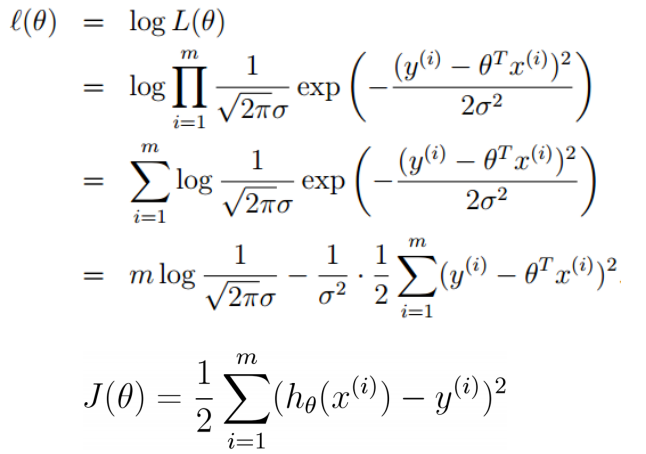
服从正态分布的原因是根据中心极限定理，在实际问题中很多现象都可以看作众多因素独立影响的综合反映，往往近似服从正态分布。如城市耗电量，就是大量用户耗电量总和，每家每户的耗电量可以看作是所有用户的平均值。

模型：

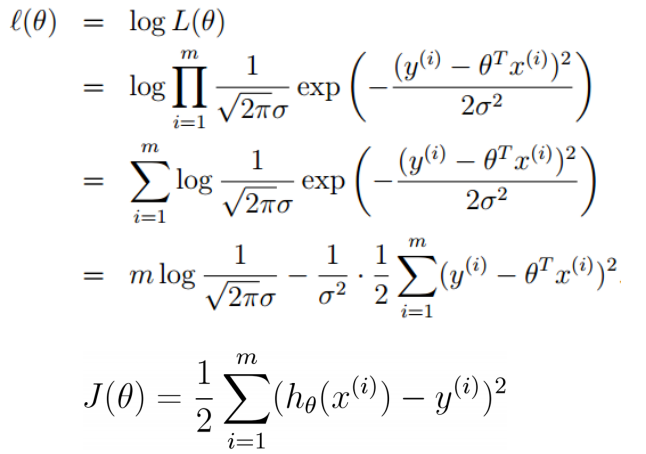
可以用正态分布表示，从而得到y的概率密度函数，使用最大似然估计（假设各变量之间是独立的）：



对数似然函数的建立：

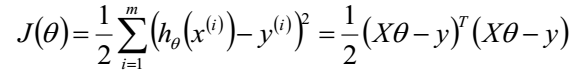


目标（损失）函数变为：求J的最小值

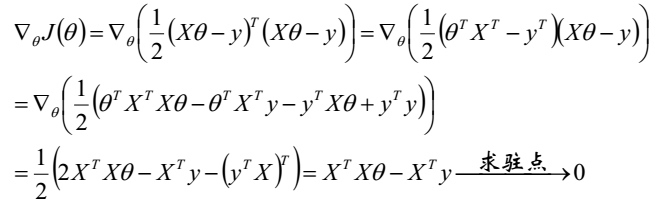


求J的最小值，也就是求解参数theta的最优解，求解出来之后带入到模型中就得到了最终想要的模型。

目标函数形式：



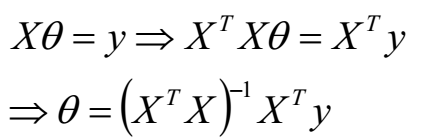
求梯度：



参数theta：

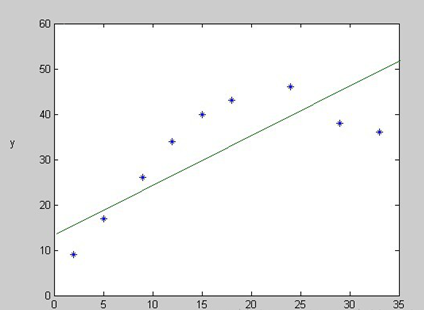


方便记忆的方式：（仅用于计算，一般x\*theta+lambda=y，lambda是误差，实际值和误差值不可能完全相等）



##### （8）局部加权回归

当目标假设是线性模型时，使用线性回归自然能够很好拟合数据，但如果目标假设不是线性模型，比如一个忽上忽下的函数，这时用线性模型就拟合的很差。比如下面坐标表示的情形，用一条直线来拟合图上的点显然时不合适的。



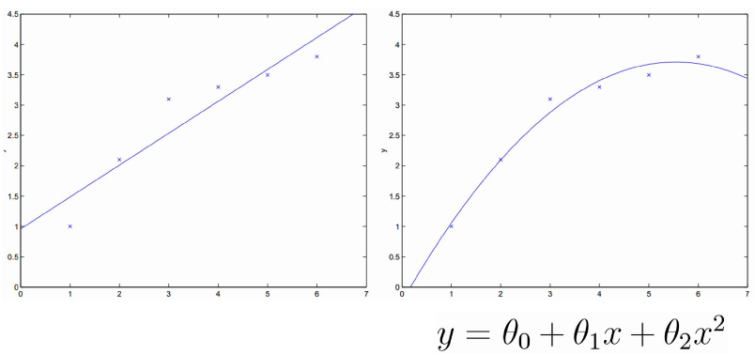
这种情况叫欠拟合（underfitting）

方法1：通过增大模型的复杂度，可以用一个二次函数做拟合

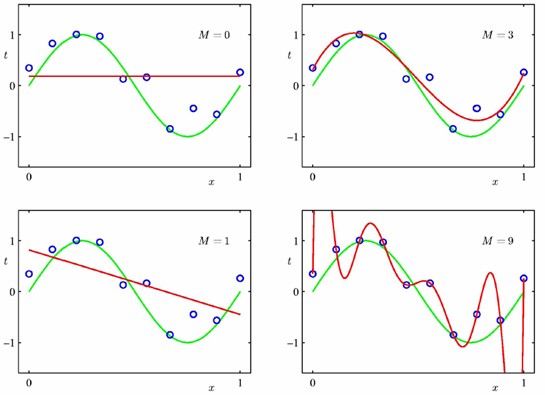
方法2：局部加权线性回归

**方法1理解：**

通常来说，对待这种情况，特征的选择可能会非常重要，比如，对于下面的图，如果加特征：x的平方或者比如加sin(x)可能能够实现很好的拟合这些点。多项式拟合能拟合所有数据（泰勒公式），但是在预测新样本的时候表现的又会变得很差很糟糕，因为它导致数据的过拟合（overfitting），不符合数据的真实的模型。



**回顾过拟合与欠拟合**

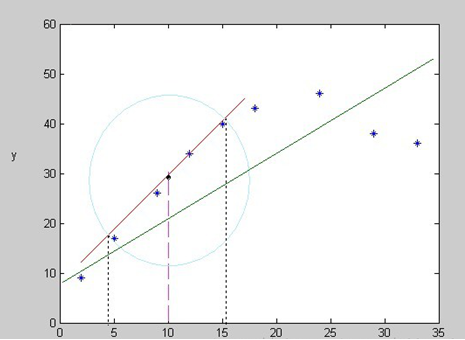


上图给出了M=0，M=1，M=3，M=9时的多项式函数拟合的情况，其中绿色曲线为真模型，红色为预测模型。

**2.采用局部加权解决模型过拟合**

针对过拟合的情况，引入局部加权方式解决。

局部加权线性回归(locally weighted linear regression)也就是在“局部”采用线性回归。



对上图的说明：假如给定x=10，要预测对应的y值，如果用线性拟合的方法，得到的是图中的绿线，有很多点没有在绿线上，误差较大。

但只考虑两条虚线之间，也就是圆圈之内的四个点进行线性拟合，那么得到的是那条红线，对应的点在这条红线上相对的y值就是预测值，从直观上来看就可以认为预测会更准确。

在待测点附近小子集上以最小均方误差来进行普通线性回归，我们使用 实现上述加权方案。

如何理解？为了实现小子集的线性回归，只取这一小部分进行线性回归。怎么只取这一小部分数据呢？这里就是通过控制，在待测点附近取1，较远的地方取0，那么就能够实现带测点附近小子集的线性回归。

当然上述我们并不严格取1或0，我们可以用近似的1和0来近似来代替。是权值，它的作用是在于**根据要预测的点与数据集中的点距离来为数据集中的点赋予权值**，当某点距离待预测点较大的时候，其权重较小，否则较大。

注意：w权值通常去的是距离分之1.

**3.非参数学习方法方式总结**

局部加权线性回归（Locally Weight Linear Regression，WLR）是一种非参数学习方法，为什么局部加权回归又叫非参数学习方法呢？

参数学习方法是这样一种方法：在训练完成数据之后得到一系列的训练参数，然后根据训练参数来预测新样本的值，这是不在依赖之前的训练数据了，参数值是确定的。而非参数学习方法是这样的一种算法：在预测新样本的时候每次都会重新训练数据得到最新的参数值，也就是每次预测新样本都会依赖训练数据集合，所以每次得到的参数是不确定的。

##### （9）正则化

模型选择的典型方法就是正则化，正则化的一般形式如下：



其中，第一项是经验风险，也就是常见的平方差损失函数，第二项是正则项，正则化项可以取不同的形式，例如，正则化项可以使模型参数的范数。在回归问题中，损失函数是平方损失，正则化项可以使参数向量的L2范数或者L1范数：





使用正则化罚项可以对不重要的属性或影响模型准确度的属性进行惩罚。

##### （10）L1和L2正则的区别，如何选择L1和L2正则？

三者的概念：

L0正则化的值是模型参数中非零参数的个数。



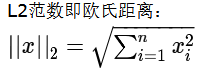
也就是如果我们使用L0范数，即希望w的大部分元素都是0。 （w是稀疏的）所以可以用于ML中做稀疏编码，特征选择。通过最小化L0范数，来寻找最少最优的稀疏特征项。但不幸的是，L0范数的最优化问题是一个NP hard问题，而且理论上有证明，L1范数是L0范数的最优凸近似，因此通常使用L1范数来代替。因此很少用L0正则化，但是在面试中会常用到。

L1正则化表示各个参数绝对值之和。



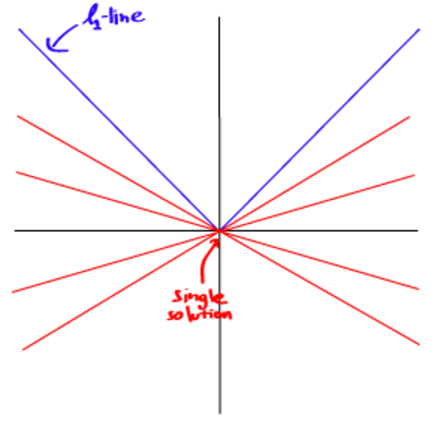
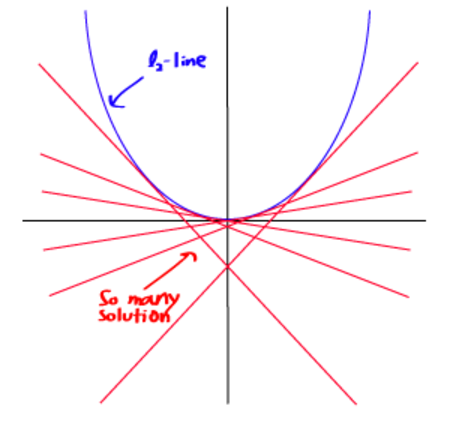
L1范数的解通常是稀疏性的，倾向于选择数目较少的一些非常大的值或者数目较多的insignificant的小值。

L2正则化标识各个参数的平方的和的开方值。



L2范数越小，可以使得w的每个元素都很小，接近于0，但L1范数不同的是他不会让它等于0而是接近于0.

如下图为：L1范数与L2范数的比较

但由于L1范数并没有平滑的函数表示，起初L1最优化问题解决起来非常困难，但随着计算机技术的到来，利用很多凸优化算法使得L1最优化成为可能。

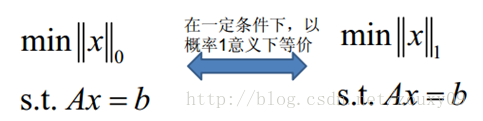
##### （11）正则项三者比较

L1 Norm 和L2 Norm的区别（核心：L2对大数，对outlier更敏感！）：

L1优点是能够获得sparse（稀疏）模型，对于large-scale的问题来说这一点很重要，因为可以减少存储空间。缺点是加入L1后目标函数在原点不可导，需要做特殊处理。  
 L2优点是实现简单，能够起到正则化的作用。缺点就是L1的优点：无法获得sparse模型。实际上L1也是一种妥协的做法，要获得真正sparse的模型，要用L0正则化。  
 L0范数是指向量中非0的元素的个数。如果我们用L0范数来规则化一个参数矩阵W的话，就是希望W的大部分元素都是0（参数W是稀疏的）。

L1范数是指向量中各个元素绝对值之和。

既然L0可以实现稀疏，为什么不用L0，而要用L1呢？**个人理解一是因为L0范数很难优化求解（NP难问题），二是L1范数是L0范数的最优凸近似，而且它比L0范数要容易优化求解。**所以大家才把目光和万千宠爱转于L1范数。



**（总结）**来个一句话总结：L1范数和L0范数可以实现稀疏，L1因具有比L0更好的优化求解特性而被广泛应用。

除了L1范数，还有一种更受宠幸的规则化范数是L2范数: ||W||2。它也不逊于L1范数，它有两个美称，在回归里面，有人把有它的回归叫“岭回归”（Ridge Regression），有人也叫它“权值衰减weight decay”。这用的很多吧，因为它的强大功效是改善机器学习里面一个非常重要的问题：过拟合。

#### 2.1.2SparkMllib岭回归

**岭回归是基于L2罚项的模型，只是在最小二乘代价函数中加入权重的平方和。**

线性回归的目标函数为：



岭回归的目标函数：



通过增加超参数的值，我们可以增加正则化的强度，同时，也就降低了权重对模型的影响。一定要注意：正则化项不影响截距

**补充：**如果数据的特征比样本点多的场景我们怎么办？是否还可以用线性回归和局部加权回归方法呢？答案是否定的，因为线性回归和局部加权回归是需要存在的前提下。而当数据的特征数比样本点多的时候我们无法保证存在，因此，为了达到进行回归预测的目的，我们**引入岭回归**，即在原目标函数的基础上加入一个，使得矩阵变成非奇异矩阵，进而能对求逆。其中I是一个m\*m的单位矩阵，对角上元素全为1，其他元素全为0，而是自定义的数值，在这种情况下回归系数的计算公式将变成：



##### （1）岭回归的目标函数

接下来探索下岭回归的目标函数。

大家还记得，线性回归的目标函数为：



我们优化线性回归的目标函数为：



为了防止过拟合，增加正则化项，目标函数变为



上述目标函数等价于





对其进行求导，得到：



令导数为0，得



上面得到的就是岭回归公式。

为了实现岭回归和缩减技术，首先需要对特征进行标准化处理，在使用上述公式进行岭回归求解。

常用得标准化方法，公式如下，样本值减去均值除以方差值



##### （2）总结

岭回归最先是用来处理**特征数大于样本数**得情况，现在也用于在估计中加入偏差，从而得到更好的估计，即控制模型参数，结构风险最小化。这里通过引入来限制了所有w之和，通过引入该惩罚项，能够减少不重要得参数，这个技术在统计学中也叫做缩减。

岭回归使用了单位矩阵乘以常量，我们观察其中得单位矩阵I，我们看到值1贯穿整个对角线，其余元素全是0.形象地，在0构成得平面上有一条1组成得“岭”，这就是岭回归中得“岭”得由来。

##### （3）岭回归优点：

1. 在特征数M大于样本数N时，不可逆，故不能直接用LR，而岭回归就可以。
2. 通过引入惩罚项，防止过拟合。

#### 2.1.3 Lasso回归和弹性网络(ElasticNet)

##### （1）Lasso回归

对于稀疏数据训练的模型，还有另一种解决方案，即LASSO，基于正则化项的强度，某些权重可以变为0，这也使得Lasso称为监督学习的一种特征选择技术。

Lasso的正则化项时L1正则，目标函数为



**注：**

Lasso的正则化项时L1正则，目标函数为



等价于



这个细微的变化，加大增加了计算复杂度，因为其不可直接求导。

Lasso回归系数：

不同于岭回归系数：

因为使用普通的最小二乘法回归在两个或更多的特征相关时，可能会得出一个很大的正系数和一个很大的负系数。可以参考岭回归图中当参数足够小的时候，岭回归变化不大，但是使用Lasso回归一些系数会因此被迫缩减到0，这个特征能够使得我们很好的理解数据。在**Lasso回归中可以避免系数过大过小的问题。**

针对不可求导的问题，如果要在新的约束条件下解出回归系数，需要使用二次规划算法，为此我们讲解一种更为简便的算法解决该问题，该方法为前向逐步回归。

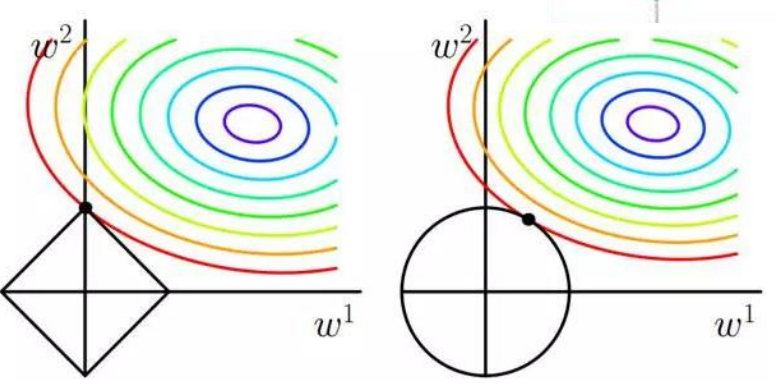
##### （2）弹性网络

弹性网络是介于岭回归和LASSO之间的一个折中，其中包含了一个用于稀疏化的L1罚项，以及一个消除Lasso限制（比如筛选变量的数量）的L2罚项。



##### （3）总结

LASSO相比Ridge回归可以进行特征选择，如下图。

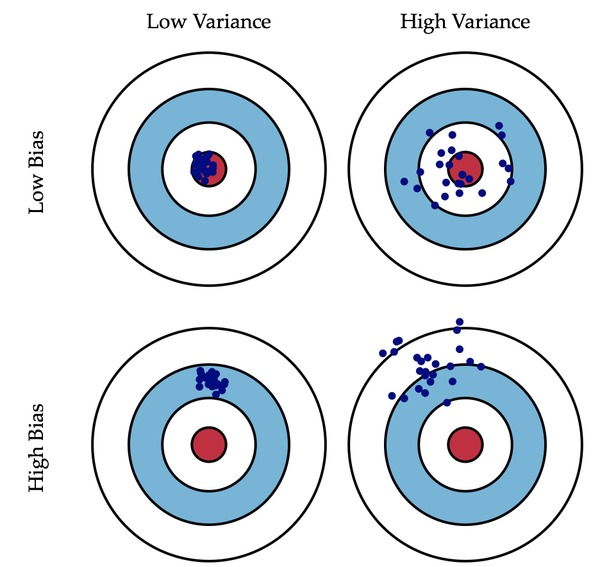


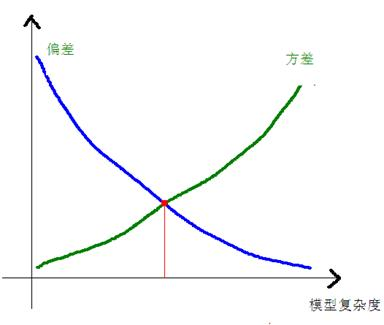
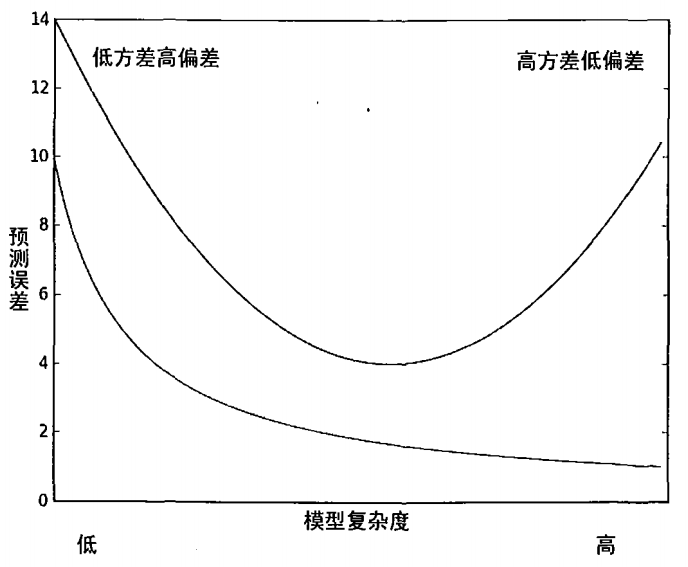
使用岭回归和Lasso回归，都对系数做了限制，模型也就增加了模型的偏差bias，与此同时却减少了模型的方差。下面我们权衡数据的偏差和方差。

#### 2.1.4权衡偏差与方差

偏差：描述的是预测值（估计值）的期望与真实值之间的差距。偏差越大，越偏离真实数据，如下图第二行所示。

方差：描述的是预测值的变化范围，离散程度，也就是离其期望值的距离。方差越大，数据的分布越分散，如下图右列所示。





利用前面的岭回归和Lasso回归可以将一些系数缩减为0，这时是增加了模型偏差，但是减少了模型的复杂，试想有些特征的系数被缩减为0，该特征就不会在线性拟合的时候起作用，就达到了属性约简，减少模型复杂度的目的。图中左侧是参数缩减过于严厉的结果，而右侧是无缩减的效果。属性无缩减的时候，模型的复杂度就比较搞了。

偏差方差折中与测试误差及训练误差的关系。右侧的曲线就是测试误差，在中间部分最低。为了做出最好的预测，我们应该调整模型复杂度（属性约简）来达到测试误差的最小值。

#### 2.1.5SparkMl回归问题实战

采用SparkMl完成线性回归预测：

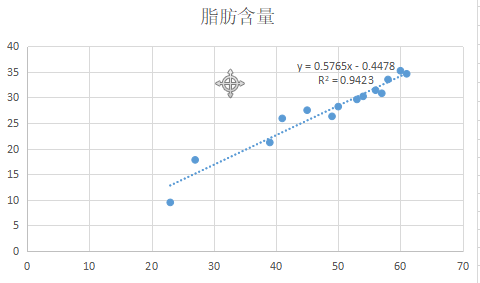
##### （1）脂肪含量回归预测分析

首先分析，简单线性回归案例，随着年龄的增加脂肪含量增加的变化曲线。

数据如下：

|  |  |
| --- | --- |
| 年龄 | 脂肪含量 |
| 23 | 9.5 |
| 27 | 17.8 |
| 39 | 21.2 |
| 41 | 25.9 |
| 45 | 27.5 |
| 49 | 26.3 |
| 50 | 28.2 |
| 53 | 29.6 |
| 54 | 30.2 |
| 56 | 31.4 |
| 57 | 30.8 |
| 58 | 33.5 |
| 60 | 35.2 |
| 61 | 34.6 |

脂肪含量曲线及R2系数



代码部分：

import org.apache.spark.ml.linalg.Vectors

import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}

import org.apache.spark.sql.SparkSession

object testDemo2 {

def main(args: Array[String]): Unit = {

val spark: SparkSession = SparkSession.builder().master("local[\*]").appName("traintestSplitTest").getOrCreate()

spark.sparkContext.setLogLevel("WARN")

val data = spark.createDataFrame(Seq(

(9.5, Vectors.dense(23)),

(17.8, Vectors.dense(27)),

(21.2, Vectors.dense(39)),

(25.9, Vectors.dense(41)),

(27.5, Vectors.dense(45)),

(26.3, Vectors.dense(49)),

(28.2, Vectors.dense(50)),

(29.6, Vectors.dense(53)),

(30.2, Vectors.dense(54)),

(31.4, Vectors.dense(56)),

(30.8, Vectors.dense(57)),

(33.5, Vectors.dense(58)),

(35.2, Vectors.dense(60)),

(34.6, Vectors.dense(61))

)).toDF("label", "features")

//1-数据切分

// val Array(train, test): Array[Dataset[Row]] = data.randomSplit(Array(0.9, 0.1), seed = 120L)

//2-训练模型

val lr: LinearRegression = new LinearRegression()

val lrModel: LinearRegressionModel = lr.fit(data)

//3- Print the coefficients and intercept for linear regression

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// 4-Summarize the model over the training set and print out some metrics

val trainingSummary = lrModel.summary

println(s"numIterations: ${trainingSummary.totalIterations}")

println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")

trainingSummary.residuals.show()

println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")

println(s"r2: ${trainingSummary.r2}")

// Coefficients: [0.5764772505370067] Intercept: -0.44779925795753567

// numIterations: 1

// objectiveHistory: [0.0]

// +--------------------+

// | residuals|

// +--------------------+

// | -3.311177504393619|

// | 2.682913493458356|

// | -0.834813512985729|

// | 2.7122319859402566|

// | 2.0063229837922307|

// | -1.4995860183557923|

// |-0.17606326889280055|

// | -0.5054950205038189|

// |-0.48197227104082785|

// | -0.4349267721148422|

// | -1.6114040226518505|

// | 0.5121187268111456|

// | 1.0591642257371348|

// |-0.11731302479987704|

// +--------------------+

//

// RMSE: 1.629890205389517

// r2: 0.9423379190667397

}

}

##### （2）运输时间预测分析

分析货运多元线性回归案例，随着运输时间和运输次数的增加，运输时间的变化曲线。

|  |  |  |  |
| --- | --- | --- | --- |
| **Driving**  **Assignment** | **X1=Miles**  **Traveled** | **X2=Number of Deliveries** | **Y= Travel Time (Hours)** |
| 1 | 100 | 4 | 9.3 |
| 2 | 50 | 3 | 4.8 |
| 3 | 100 | 4 | 8.9 |
| 4 | 100 | 2 | 6.5 |
| 5 | 50 | 2 | 4.2 |
| 6 | 80 | 2 | 6.2 |
| 7 | 75 | 3 | 7.4 |
| 8 | 65 | 4 | 6.0 |
| 9 | 90 | 3 | 7.6 |
| 10 | 90 | 2 | 6.1 |

Time = -0.869 + 0.0611 Miles + 0.923 Deliveries

代码实战：

import org.apache.spark.ml.linalg.Vectors

import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}

import org.apache.spark.sql.SparkSession

object testDemo2 {

def main(args: Array[String]): Unit = {

val spark: SparkSession = SparkSession.builder().master("local[\*]").appName("traintestSplitTest").getOrCreate()

spark.sparkContext.setLogLevel("WARN")

val data = spark.createDataFrame(Seq(

(9.3, Vectors.dense(100, 4)),

(4.8, Vectors.dense(50, 3)),

(8.9, Vectors.dense(100, 4)),

(6.5, Vectors.dense(100, 2)),

(4.2, Vectors.dense(50, 2)),

(6.2, Vectors.dense(80, 2)),

(7.4, Vectors.dense(75, 3)),

(6.0, Vectors.dense(65, 4)),

(7.6, Vectors.dense(90, 3)),

(6.1, Vectors.dense(90, 2))

)).toDF("label", "features")

//1-数据切分

// val Array(train, test): Array[Dataset[Row]] = data.randomSplit(Array(0.9, 0.1), seed = 120L)

//2-训练模型

val lr: LinearRegression = new LinearRegression().setMaxIter(50)

val lrModel: LinearRegressionModel = lr.fit(data)

//3- Print the coefficients and intercept for linear regression

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// 4-Summarize the model over the training set and print out some metrics

val trainingSummary = lrModel.summary

println(s"numIterations: ${trainingSummary.totalIterations}")

println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")

trainingSummary.residuals.show()

println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")

println(s"r2: ${trainingSummary.r2}")

// Coefficients: [0.06113459879206173,0.9234253666954294] Intercept: -0.8687014667816825

// numIterations: 1

// objectiveHistory: [0.0]

// +--------------------+

// | residuals|

// +--------------------+

// | 0.3615401207937943|

// |-0.15830457290769218|

// |-0.03845987920620608|

// | -0.5916091458153492|

// | 0.1651207937877377|

// | 0.3310828300258857|

// | 0.9133304572907646|

// | -0.7987489214840471|

// | 0.1963114754098383|

// | -0.3802631578947322|

// +--------------------+

//

// RMSE: 0.479525128201181

// r2: 0.9037889754910632

}

}

##### （3）SparkMl实战Libsvm数据线性回归实战

import org.apache.spark.ml.regression.LinearRegression

//1- Load training data

val training = spark.read.format("libsvm")

.load("data/mllib/sample\_linear\_regression\_data.txt")

val lr = new LinearRegression()

.setMaxIter(10)

.setRegParam(0.3)

.setElasticNetParam(0.8)

// 2-Fit the model

val lrModel = lr.fit(training)

//3- Print the coefficients and intercept for linear regression

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// 4-Summarize the model over the training set and print out some metrics

val trainingSummary = lrModel.summary

println(s"numIterations: ${trainingSummary.totalIterations}")

println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")

trainingSummary.residuals.show()

println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")

println(s"r2: ${trainingSummary.r2}")

#### 2.1.6SparkMllib回归问题实战

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.regression.LinearRegressionModel

import org.apache.spark.mllib.regression.LinearRegressionWithSGD

//1- Load and parse the data

val data = sc.textFile("data/mllib/ridge-data/lpsa.data")

val parsedData = data.map { line =>

val parts = line.split(',')

LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(\_.toDouble)))

}.cache()

// 2-Building the model

val numIterations = 100

val stepSize = 0.00000001

val model = LinearRegressionWithSGD.train(parsedData, numIterations, stepSize)

// 3-Evaluate model on training examples and compute training error

val valuesAndPreds = parsedData.map { point =>

val prediction = model.predict(point.features)

(point.label, prediction)

}

val MSE = valuesAndPreds.map{ case(v, p) => math.pow((v - p), 2) }.mean()

println("training Mean Squared Error = " + MSE)

//4- Save and load model

model.save(sc, "target/tmp/scalaLinearRegressionWithSGDModel")

val sameModel = LinearRegressionModel.load(sc, "target/tmp/scalaLinearRegressionWithSGDModel")

#### 2.1.7SparkMllib回归问题实战房价预测

数据描述：

* position;地理位置
* square;面积
* price;价格
* direction;方位
* type;类型
* name；小区名字

数据部分：

position;square;price;direction;type;name；  
 0;190;20000;0;4室2厅2卫;A;  
 0;190;20000;0;4室2厅2卫;A;  
 5;400;15000;0;4室3厅3卫;B;  
 0;500;15000;0;5室3厅2卫;C;  
 5;500;15000;0;5室3厅4卫;D;  
 1;320;15000;1;1室1厅1卫;E;  
 0;143;12000;0;3室2厅2卫;B;  
 0;200;10000;0;4室3厅2卫;D;  
 0;207;9000;0;4室3厅4卫;C;  
 0;130;8500;0;3室2厅2卫;F;  
 5;150;7000;0;3室2厅2卫;B;

代码部分：

package org.apache.spark.examples.examplesforml

import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}

import org.apache.spark.ml.regression.LinearRegression

import org.apache.spark.sql.{DataFrame, SparkSession}

import org.apache.spark.{SparkConf, SparkContext}

import scala.util.Random

/\*数据集：house.csv\*/

object Linear {

def main(args: Array[String]): Unit = {

//1-准备环境

val conf = new SparkConf().setMaster("local[\*]").setAppName("LinearregRession")

val sc = new SparkContext(conf)

val spark = SparkSession.builder().config(conf).getOrCreate()

//2-读取数据

val file = spark.read.format("csv")

.option("header", "true") //y

.option("sep", ";") //分隔符

.load("D:\\BigData\\Workspace\\SparkMllibLesson\\src\\main\\scala\\sparkmllib\_part1\\test\\HomePriceEstimator\\USA\_Housing.txt")

file.show()

import spark.implicits.\_

val random = new Random()

//3-特征选择

val data = file.select("square", "price", "type")

.map(row => (row.getAs[String](0).toDouble, row.getAs[String](1).toDouble, row.getAs[String](2).toString, random.nextDouble()))

.toDF("square", "price", "type", "rand")

.sort("rand")

data.show()

//4-特征工程- stringIndexer

val stringIndexer = new StringIndexer().setInputCol("type").setOutputCol("typeIndexer").fit(data)

val transdata = stringIndexer.transform(data)

transdata.show()

//4-特征工程- VectorAssembler

val assembler = new VectorAssembler()

.setInputCols(Array("square", "typeIndexer"))

.setOutputCol("features")

val dataset = assembler.transform(transdata)

//5-特征工程- 数据集切分

var Array(train, test) = dataset.randomSplit(Array(0.8, 0.2), 1234L)

train.show()

println(test.count())

////6建模分析-房屋的面积和房屋的价格的关系

var regression = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

val model = regression.setLabelCol("price").setFeaturesCol("features").fit(train)

model.transform(test).show()

//7-模型校验

val s = model.summary.totalIterations

println(s"iter: ${s}")

}

}

### 2.2SparkMllib逻辑回归的原理及实战

**逻辑回归虽然带有回归二字但是确是一个分类模型，在学习完线性回归以后学习逻辑回归更加简单，这点需要明确。**

分类模型，常用作二分类

结果常通过概率来表示，哪一个类别概率大就判断为哪一个分类。我们首先熟悉概率的基本知识以及对极大似然估计得到对参数的最佳估计值。

#### 2.2.1概率知识回顾

1. 定义：概率：对一个事情可能性的衡量
2. 范围：0<=P<=1
3. 计算方法：

公式法（最常用的）：

P(随机事件)=**.**其**中**P(必然事件)=1,P（不可能事件）=0；0<P(随机事件)<1.

2

3

图1

1

4

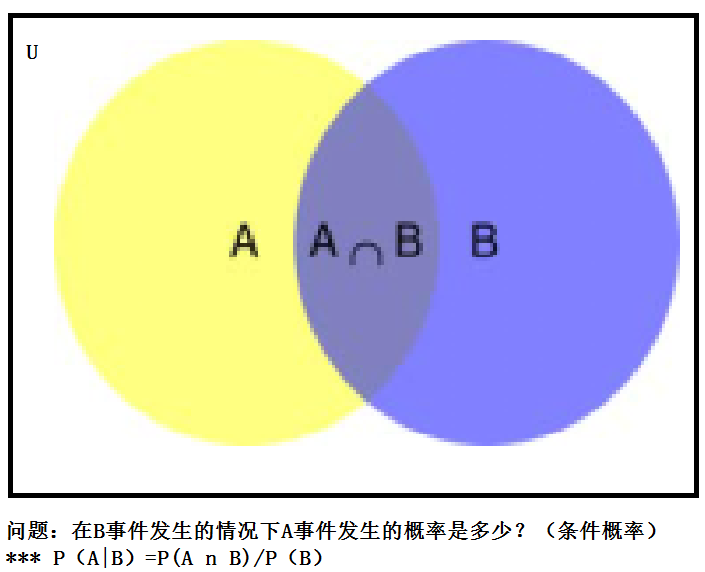
5

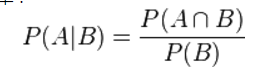
6

例：图1中每一个标有数字的方块均是可以翻动的木牌，其中只有两块木牌的背面贴有中奖标志，则随机翻动一块木牌中奖的概率为\_\_\_\_\_\_\_\_．

**解析:** 本题考查用公式法求概率,在随机翻动木牌过程中，一共有6种可能的翻牌结果,其中有2种为中奖,所以P(中奖)=.

1. 条件概率





#### 2.2.2极大似然估计

**极大似然估计是概率的log概率的最大化问题，即max(log（P）)，log函数是单调函数，通常取底数大于1的情况，所以log函数不影响原来概率大小的判断。如果取极大似然函数的相反数，得到的就是我们熟悉的损失函数cost，同时极大似然函数也对应的极小化我们的损失函数。**

##### （1）为什么要有极大似然估计？

例子：我与一位猎人一起外出打猎，一只野兔从前方穿过，只听到一声枪响，野兔应声倒下。问是谁打中的呢？

答：极有可能是猎人。

显然候选人就两个，我和猎人。若选择我，则事件发生的发生概率为0.01%，因为我不会打猎；若选择猎人，则事件发生的概率为99%，而事件已经发生，因此选择猎人更为合适。

极大似然法的基本思想在社会思维意识中常有所体现。例如某地发生了一个疑难案件，警察欲破案或民众推测嫌疑人，一般是将重点集中在作案可能性较大的可疑人身上。

极大似然估计的思想

设总体中含有待估参数w，可以取很多值。已经知道了样本观测值（例子中的兔子被猎人打死了），从w的一切可能值中（引例中是我和猎人）选出一个使该观察值出现的概率为最大的值，作为w参数的估计值，这就是极大似然估计。（顾名思义：就是看上去那个是最大可能的意思）

##### （2）极大似然估计步骤：

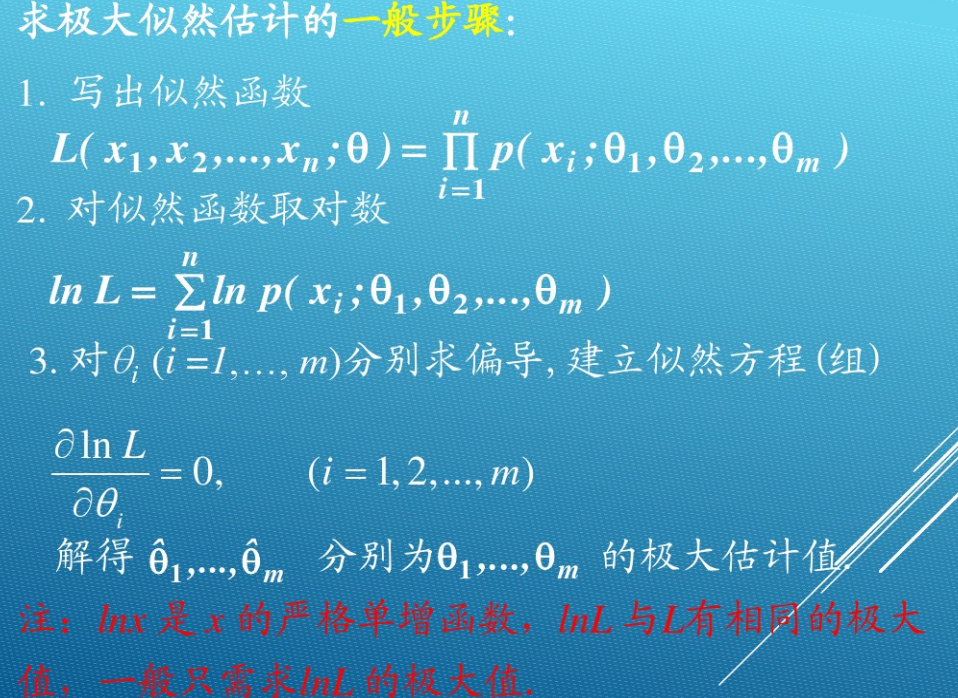
求极大似然函数估计值的一般步骤：

　　（1） 写出似然函数；

　　（2） 对似然函数取对数，并整理；

　　（3） 求导数 ；

（4） 解似然方程



极大似然估计，只是一种概率论在统计学的应用，它是参数估计的方法之一。说的是已知某个随机样本满足某种概率分布，但是其中具体的参数不清楚，参数估计就是通过若干次试验，观察其结果，利用结果推出参数的大概值。极大似然估计是建立在这样的思想上：已知某个参数能使这个样本出现的概率最大，我们当然不会再去选择其他小概率的样本，所以干脆就把这个参数作为估计的真实值。

当然极大似然估计只是一种粗略的数学期望，要知道它的误差大小还要做区间估计。

##### （3）极大似然估计的案例

设某工序生产的产品的不合格率为，抽个产品作检验，发现有个不合格，试求的极大似然估计．

**分析：**设是抽查一个产品时的不合格品个数，则服从参数为的二点分布．抽查个产品，则得样本，其观察值为，假如样本有个不合格，即表示中有个取值为１，个取值为０．按离散分布场合方法，求的极大似然估计．

**解：**（１）写出似然函数：

（２）对取对数，得对数似然函数：



（３）由于对的导数存在，故将对求导，令其为０，得似然方程：

（４）解似然方程得：

（５）经验证，在时，，这表明可使似然函数达到最大

（６）上述过程对任一样本观测值都成立，故用样本代替观察值便得的极大似然估计为：

将观察值代入，可得的极大似然估计值为：，其中．

若总体的分布中含有多个未知参数时，似然函数是这些参数的多元函数．代替方程（３），我们有方程组，由这个方程组解得分别是参数的极大似然估计值．

##### （4）无约束优化方法简介（补充）

无约束优化方法是优化技术中极为重要和基本内容之一。它不仅可以直接用来求解无约束优化问题，而且很多约束优化问题也常将其转化为无约束优化问题（比如在SVM中我们将有约束条件的最优化问题利用**拉格朗日函数**转化为无约束条件的问题，从而利用梯度下降方法等最优化理论进行求解最优解），然后用无约束优化方法来求解。

最速下降法和牛顿法是比较常见的求解无约束问题的最优化方法，这两种算法作为基本算法，在最优化方法中占有重要的地位。其中最速下降法又称梯度法，其**优点是**工作量少，存储变量较少，初始点要求不高；**缺点**是收敛慢，效率低。牛顿法的**优点是**收敛速度快；**缺点是**对初始点要求严格，方向构造困难，计算复杂且占用内存较大。

#### 2.2.3梯度下降法

**为什么使用梯度下降法？**

（1）**在机器学习的优化问题中，梯度下降法和牛顿法是常用的两种凸函数求极值的方法，他们都是为了求得目标函数的近似解。**在逻辑斯蒂回归模型的参数求解中，一般用改良的梯度下降法，也可以用牛顿法。两种方法我们在课程最后会进行比较。

（2）**梯度实际上是函数值变化最快的方向。**

比如说，你站在一个山上，梯度所指示的方向是高度变化最快的方向。你沿着这个方向走，能最快的改变（增加或是减小）你所在位置的高度，但是如果你乱走，可能走半天所在位置高度也没有变化多少。也就是说，如果你一直沿着梯度走，你就能最快的到达山的某个顶峰或低谷（偶尔会到鞍点，不过这几乎不可能）。

所以实际上，梯度下降法是用来数值搜索局部极小值或极大值的，它是实际应用中一种非常高效，高速且可靠的方法。

接下来首先了解几个数学上的概念：

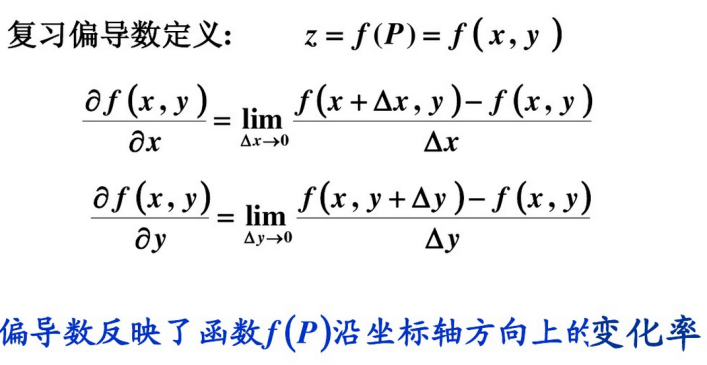
**方向导数**：是一个数；反映的是f(x,y)在P0点沿方向v的变化率。

**偏导数**：是多个数（每元有一个）；是指多元函数沿坐标轴方向的方向导数，因此二元函数就有两个偏导数。

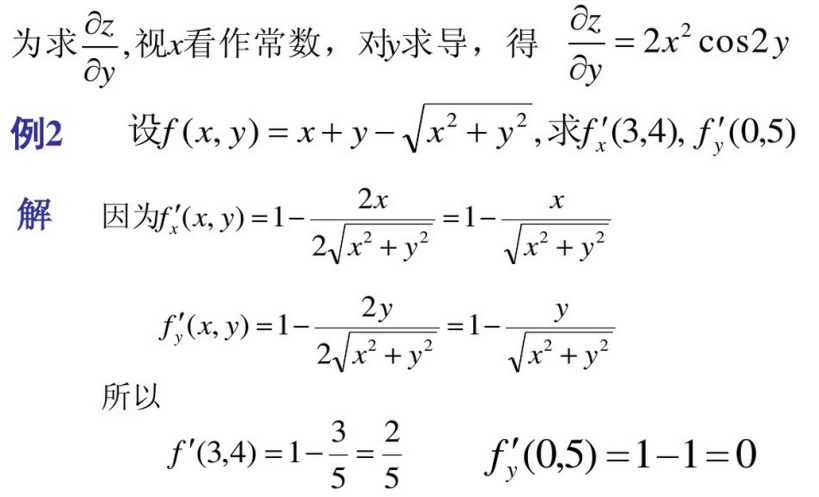
**偏导函数**：是一个函数；是一个关于点的偏导数的函数。

**梯度**：是一个向量；每个元素为函数对一元变量的偏导数；它既有大小（其大小为最大方向导数），也有方向。

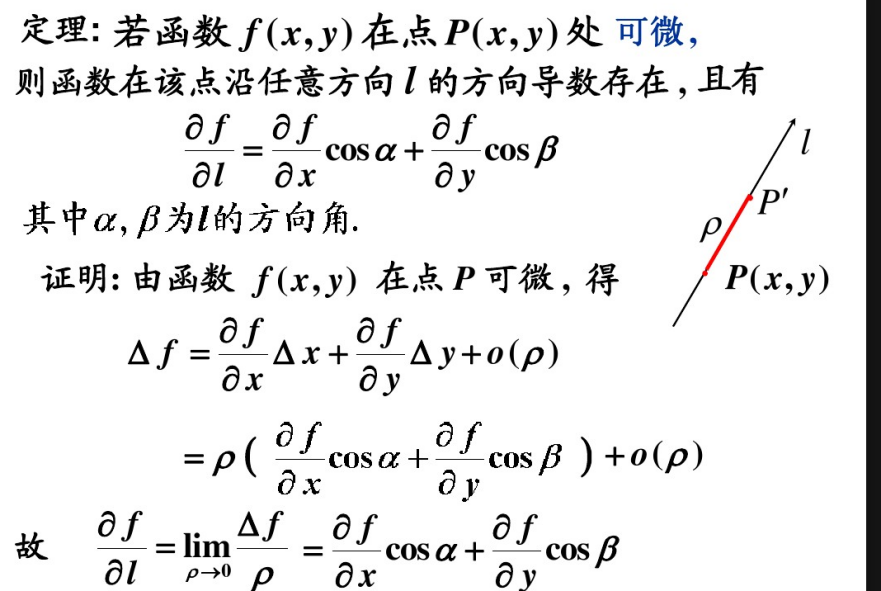
##### （1）偏导数

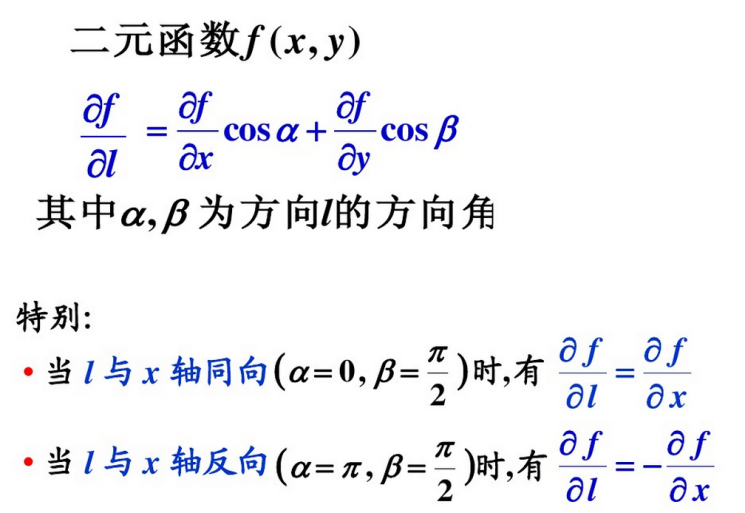


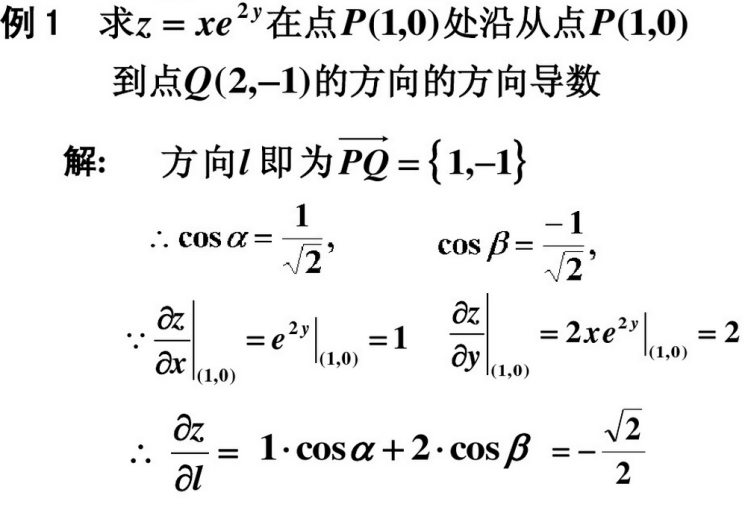




##### （2）方向导数



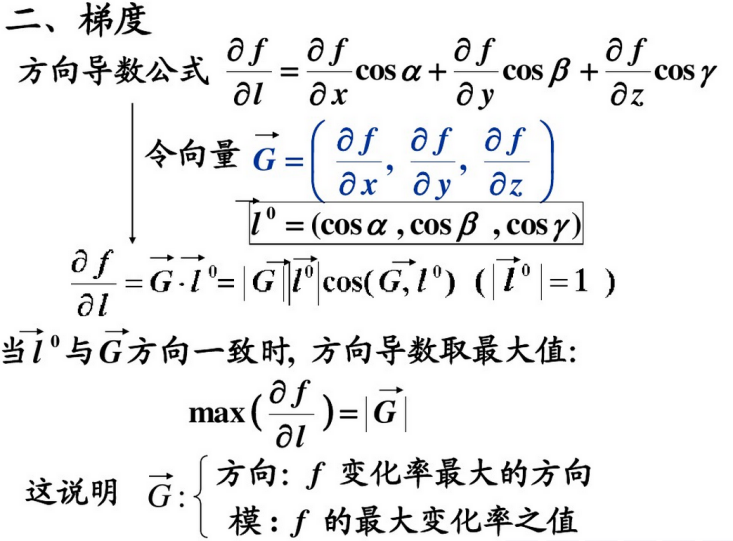


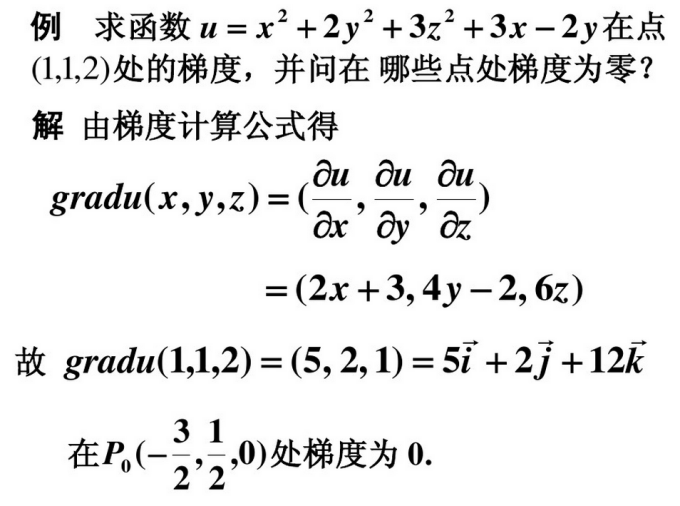


##### （3）梯度

梯度是一个向量；既有大小，也有方向。

梯度的方向是方向导数中取到最大值的方向，梯度的值是方向导数的最大值。





  函数z=f(x,y)在点P0处的梯度方向是函数变化率(即方向导数)最大的方向。

  梯度的方向就是函数f(x,y)在这点增长最快的方向，梯度的模为方向导数的最大值。

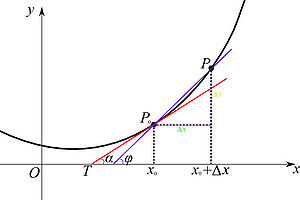
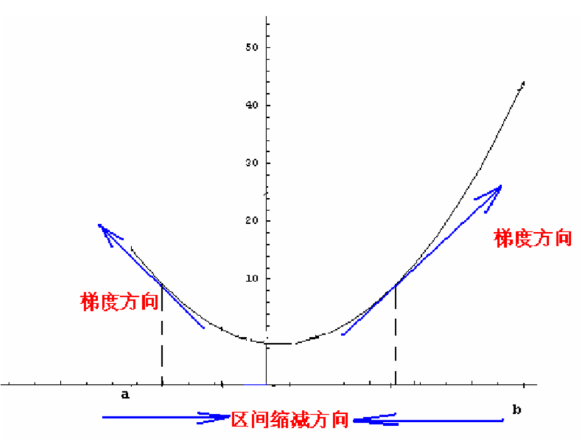
##### （4）梯度下降

我们先来玩一个游戏，假如你在一座山上，蒙着眼睛，但是你必须到达山谷中最低点的湖泊，你该怎么办？

对，梯度可以帮助你完成这个游戏。



通过上述数学上面的学习和推导，我们已经了解梯度值是方向导数的最大值，梯度的方向是方向导数取最大值的时候所对应的方向。如下图所示。我们在机器学习中频繁使用梯度的知识求解参数的最优解，这里需要目标凸函数是凸函数的限制，因此在目标函数中经常会取平方项而构成凸函数。



目标函数：



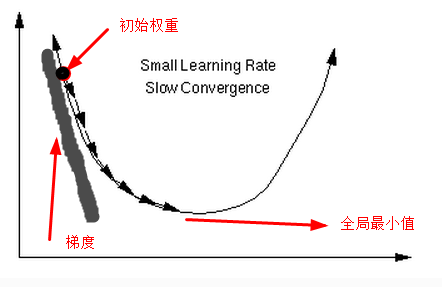
注意：

1.系数1/2只是出于方便考虑，他是我们更加容易求出梯度。

2.我们使用平方的另一个方面是相对与其他的一些函数（如阶跃函数，在x=0处不可导），上述函数在定义域内函数是处处可导的。

3.加了平方项构成凸函数，我们可以通过梯度下降优化算法来得到权重，并且最优解能使得误差函数最小。

梯度下降算法可以描述为“下山”，直到获得一个局部或者全局最小值。在每次的迭代中，根据学习速率和梯度的斜率，能够确定每次移动的步数。



由此，我们基于上述的损失函数沿着梯度方向权重更新如下：



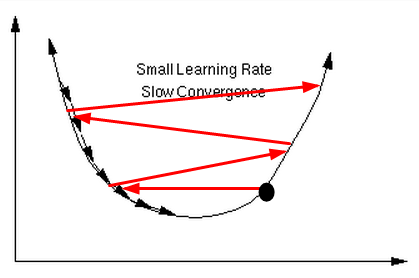
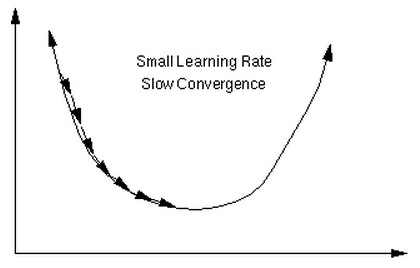
我们需要对损失函数J中的每个权重进行计算，并且把每次做出的更新累加到最后的权值w上。其中为学习率，控制每次跨越的步数。（代价函数的导数）决定朝那个方向去更新权值。



上面的梯度下降需要遍历所有的样本点，也就是说权重是基于训练集中所有的样本而完成的（而不是每一次样本的逐渐更新），这就是我们常称作的批量梯度下降法（BGD）。

**学习速率的分析：**

1. 学习速率设置的过小，使得收敛速度慢，找到全局最优解需要的时间长，计算量大。
2. 学习速率设置的过大，梯度下降算法可能会跳过全局最优解。如下图所示。



**优缺点分析：**

基于批量梯度下降算法每次每一次遍历的是全部样本，收敛速度慢。一般工程中用到它的改进版本：随机梯度下降。与批量不同的是，随机梯度下降每次只使用一个训练样本渐进的更新权重，一般该方法因为权重更新频繁，通常可以更快收敛，找到最低点或最优值。

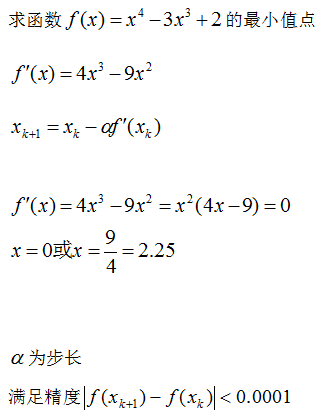
随机梯度下降特点：

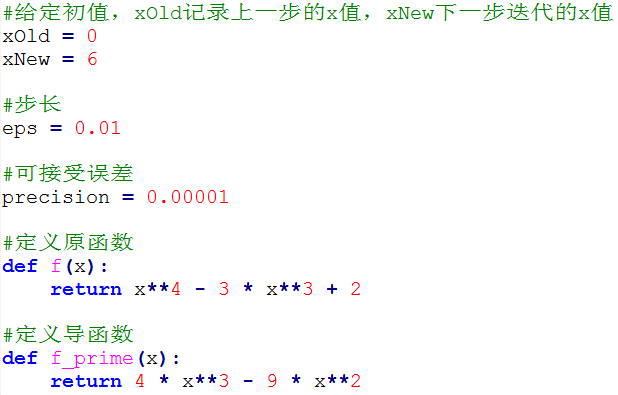
不会精确收敛到最低点，会在最低点附近徘徊

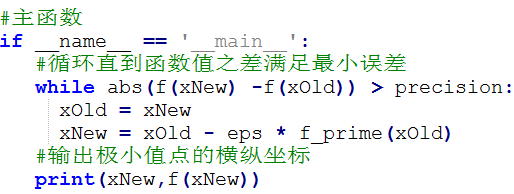
速度快

无需遍历所有样本点。

##### （5）梯度下降求解函数极值：







结果输出：



##### （6）梯度下降法总结

（1）方向导数是各个方向上的导数

（2）偏导数连续才有梯度存在

（3）梯度的方向是方向导数中取到最大值的方向，梯度的值是方向导数的最大值。

##### （7）梯度下降法求解线性回归问题参数的最优解

线性回归中利用极大似然估计和高斯分布推导最小二乘法得到了参数theta的解析解（解析解带有矩阵的逆，有时候计算复杂度很高），但是很多时候更希望有数值解，我们就可以使用梯度下降法求解。

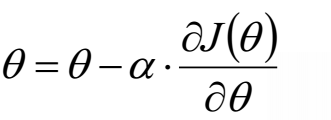
参数theta解析解：



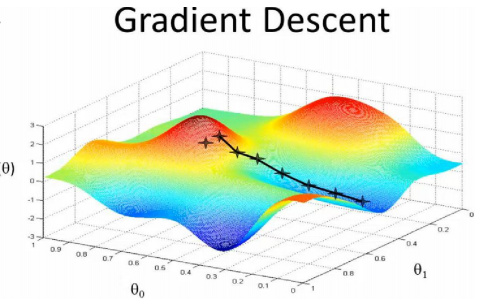
接下来我们沿着梯度的方向下降，注意：给定一个学习率a，参数要设置适当，设置太小速度太慢，设置太大可能错过最低点。

梯度下降法步骤：

* 随机初始化theta
* 沿着梯度方向迭代，更新后的theta使得J最小，a是学习率

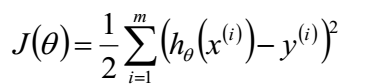


参考下面图示：

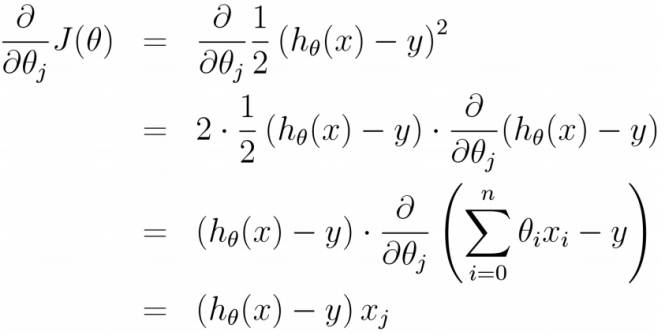


**最后，求解目标函数的参数最优解**

从新考虑J目标函数：

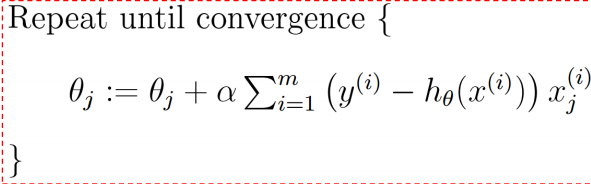


利用梯度下降法求导：

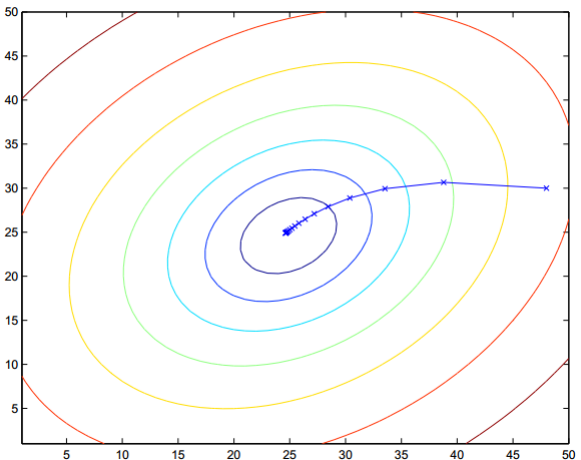


##### （8）批量梯度下降

更新规则--所有样本都参与了theta的更新和求解，这称之为批量梯度方法，批量梯度下降法可以找到**线性回归**的全局最小值（为什么？因为目标函数是凸函数，凸函数有且只有一个全局最小值），但算法本身局限在于可能存在局部最优解，但不是全局最优解。

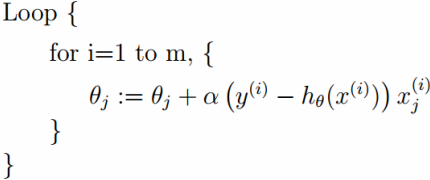


下图为批量梯度下降图示：



##### （9）随机梯度下降法SGD

（特点）更快，在线，可以跳过局部最小值，有可能找不到全局最优值，有时候会在局部最优值点发生震荡，但是一般情况下在一定位置发生震荡，认为模型收敛了。SGD比BGD更能收敛到全局最优值。

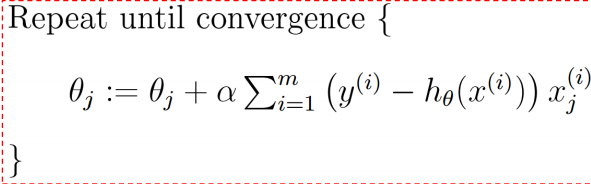
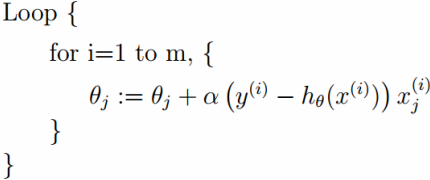


##### （10）Mini-batch

上述两种梯度下降的折中

通常将Mini-batch SGD也称之为SGD

如果不是每拿到一个样本更改梯度，而是若干样本的平均梯度作为更新方向，则是mini-batch梯度下降算法。

#### 2.2.4牛顿法

牛顿法和我们之前听过的梯度下降法都是属于最优化理论部分，最常见的情形就是利用目标函数的导数通过多次迭代来求解无条件约束最优化问题。实现简单，coding 方便，是训练模型的必备利器之一。

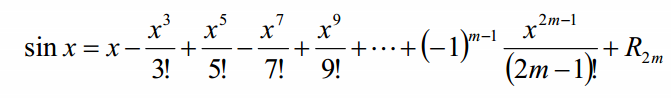
##### （1）泰勒公式（讲解）

泰勒公式

迈克劳林公式

下面是两个例子：





##### （2）求解方程根

不是所有的方程都有求根公式，或者求根公式很复杂，导致求解困难，可以利用牛顿法，可以迭代求解。

利用泰勒公式，在处展开到一阶



求解方程，即

解得：

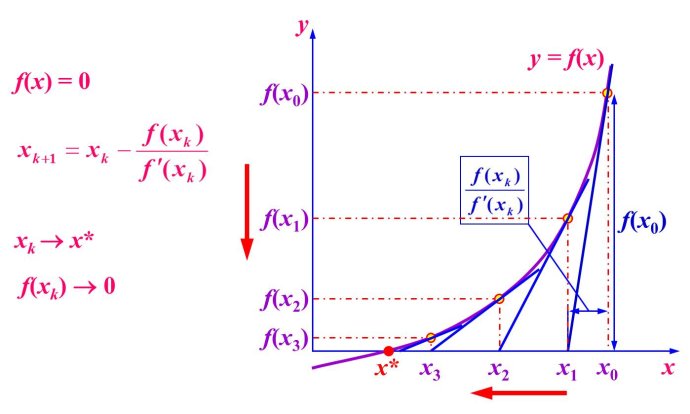


利用泰勒公式的一阶展开，处并不是完全相等，而是近似相等，这里求得的并不能让，只能说比更接近，于是继续迭代求解。可以推出：



通过迭代，上面式子一定能在的时候收敛。也成为牛顿法的一次迭代（收敛）。

通过下图理解迭代求解的过程：

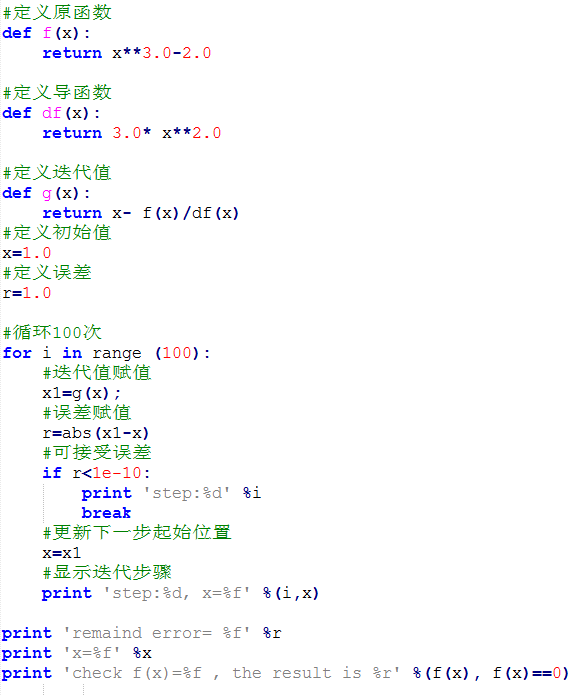


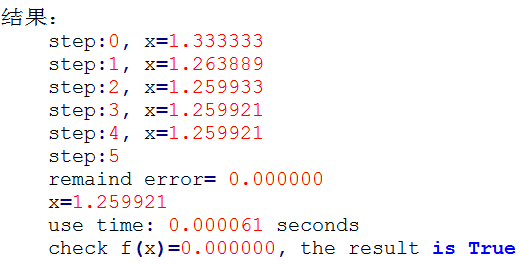
##### （3）实战牛顿法求解方程

例：利用牛顿法求方程，解为

利用以下公式求解：







##### （4）牛顿迭代法

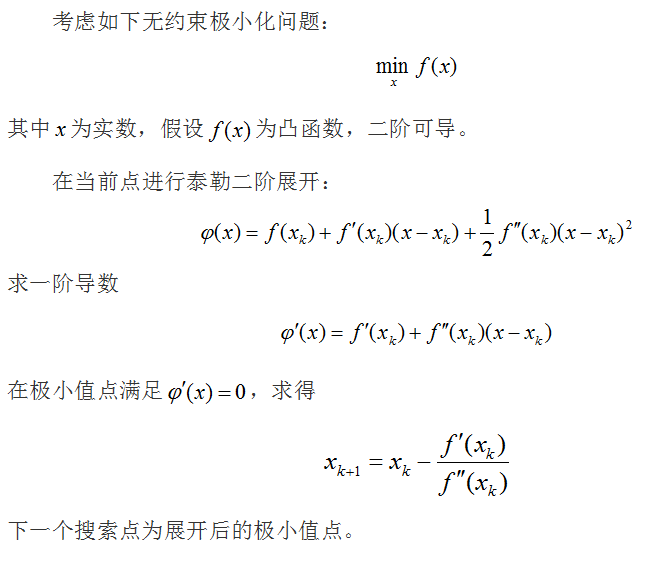
(1)一次迭代



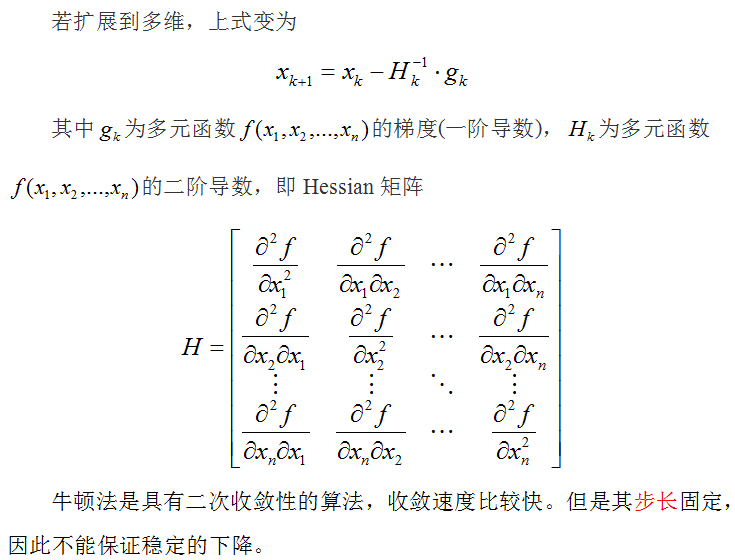
(2)二次迭代（收敛）



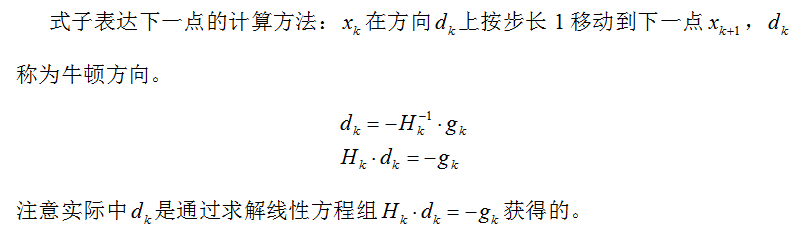
（3）具体推导如下过程：

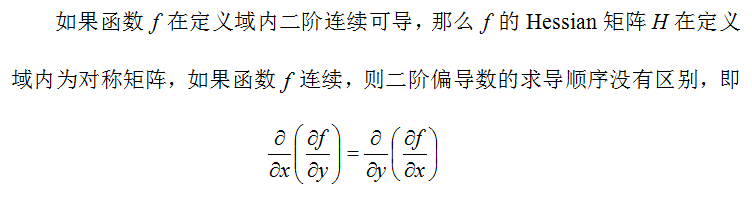


##### （5）多维特征的牛顿迭代法





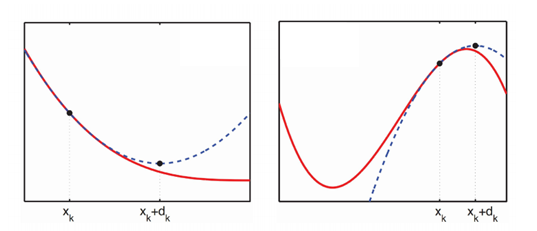


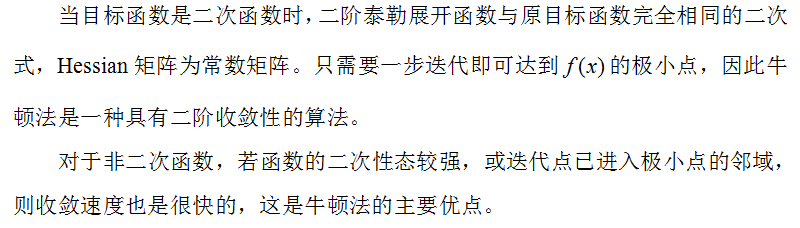


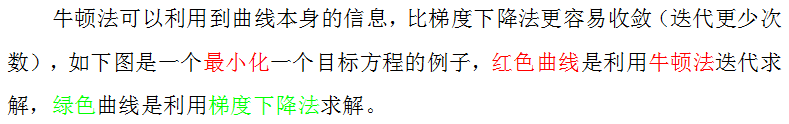


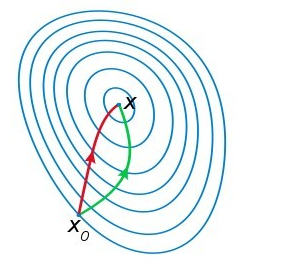
H阵为对称阵。













##### （6）两个改进方法

**阻尼牛顿法**：在牛顿方向上附加了步长因子，每次调整时会在搜索空间，在该方向找到最优步长，然后调整。

**拟牛顿法**：对或取近似值，可减少计算量。

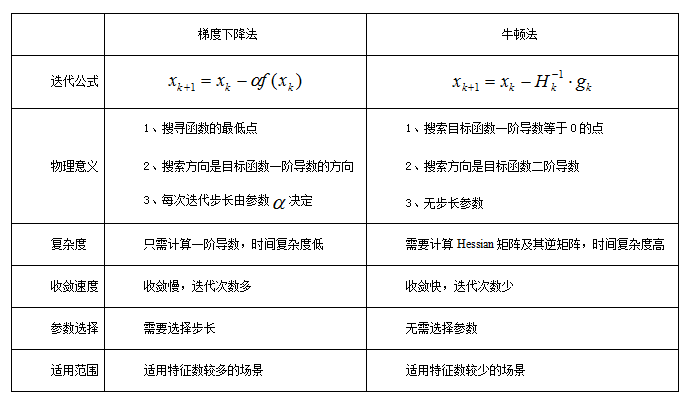
##### （7）牛顿法求解实例

例 用牛顿法求解：

牛顿法公式



##### （8）梯度下降法与牛顿法的比较



牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。如果更通俗地说的话，比如你想找一条最短的路径走到一个盆地的最底部，梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。

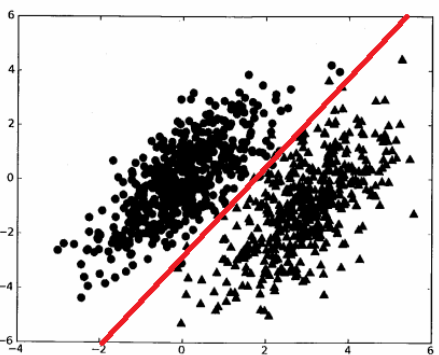
#### 2.2.5Logistic Regression (逻辑回归)

**分类问题的首选算法。（类别+概率）**

应用场景：对于许多应用实践来说，我们不但对类标预测感兴趣，而且对事件属于某一类别的概率进行预测也非常有用。例如，将逻辑斯特回归应用于天气预报，不仅要预测某天是否会下雨，还要推测下雨有多大的可能性。同样，逻辑斯特回归还可用于预测出现某些症状的情况下，患者患有某种疾病的可能性，这也是逻辑斯特回归在医疗领域得到了很广泛的研究。

##### （1）引入例子

如果y是连续值，我们可以采用回归分析，但如果y是离散值，使用logistic回归解决回归问题。

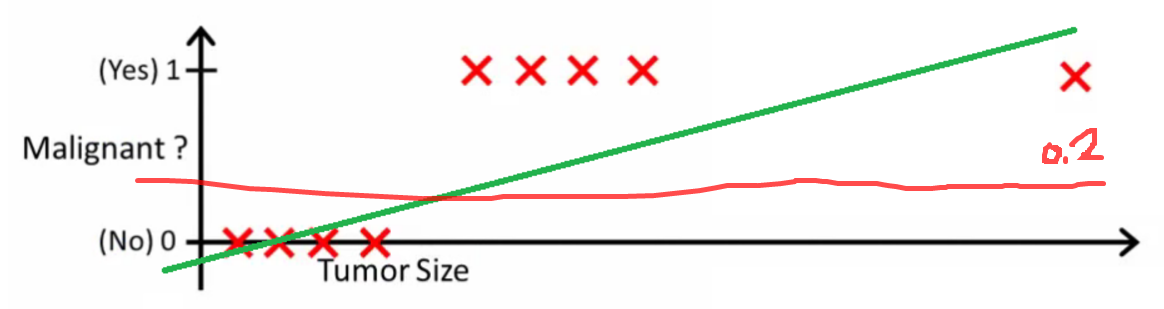




设置h(x) > 0.5

通过上图中的分割线，如果新样本是一个比较特别的点，如下图最右侧的X，新样本应该预测为1，但根据上面的分割线却被分割到了第0类，因此，对于分类的问题用先行回顾较差。

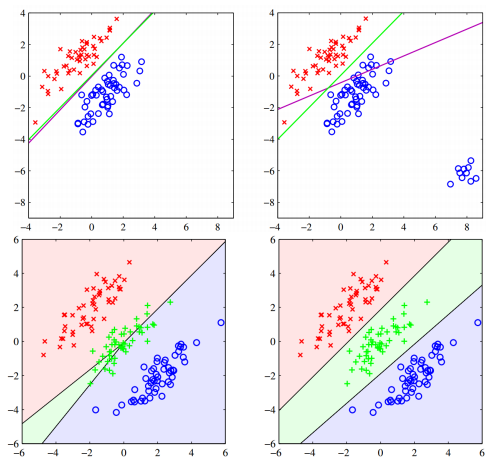
这种分类实际上可以可以设定一个y值，如果h(x) > 0.5是第一类，h(x) <0.5是第0类，下图可能会设置y=0.2可以更好的分类。

 设置h(x) > 0.2

**看图观察：**

图1是线性回归，图2是logistic回归（效果好）

图3是线性回归处理多类别问题，图4是softmax处理多类别问题（效果好）



##### （2）函数模型

**（1）逻辑斯特函数的由来**

假设一事件发生的概率为P，则不发生的概率为1-P，我们把发生概率/不发生概率称之为发生的概率比，数学公式表示为：



更进一步我们定义logit函数，它是概率比的对数函数（log-odds）



Logit函数输入值范围介于[0,1]之间，它能将输入转换到整个实数范围内。

对logit函数求反函数，我们将logit的反函数叫做logistic函数（课堂推到如何求解反函数，属于高中阶段的知识）



**（2）进一步探究**

         测试数据为

         要学习的参数为：

模型的线性表示：（样本特征与权重的线性组合）



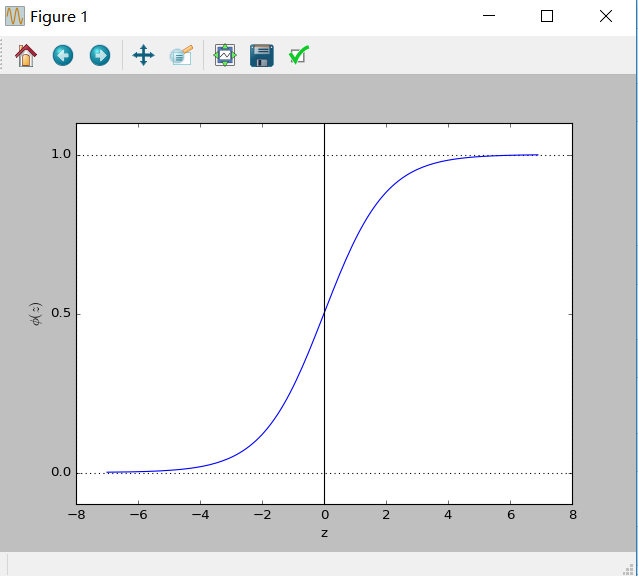
向量表示：



 处理二值数据，引入Sigmoid函数时曲线平滑化



该函数的图像如下图：



对图像的理解：sidmod函数以实数值作为输入并将其反射到[0，1]区间，拐点在y=0.5地方。

##### （3） Sigmod函数绘图实战

需求：绘制[-7，7]的sigmod函数图像

import matplotlib.pyplot as plt

import numpy as np

def sigmod(z):

return 1.0/(1.0+np.exp(-z))

z=np.arange(-7,7,0.1)

phi\_z=sigmod(z)

plt.plot(z,phi\_z)

plt.axvline(0.0,color='k')

plt.axhspan(0.0,1.0,facecolor='1.0',alpha=1.0,ls='dotted')

plt.yticks([0.0,0.5,1.0])

plt.ylim(-0.1,1.1)

plt.xlabel('z')

plt.ylabel('$\phi (z)$')

plt.show()

##### （4）函数表示

预测函数：



  用概率表示：

      正例(y=1)：



      反例(y=0):



用一个函数综合表述上面两个函数：



##### （5）似然函数

逻辑斯特回归似然函数一般定义为：

注意：这里数据集中的每个样本都是相互独立的



注意：

接下来的目标变为了：找参数的极大似然估计

（1）目标函数加log处理



1. **目标函数求导，并利用梯度上升算法求解参数**

第一种求导思路：





求导的第二种思路：（建议这种方式）



1. **梯度上升算法求解参数**

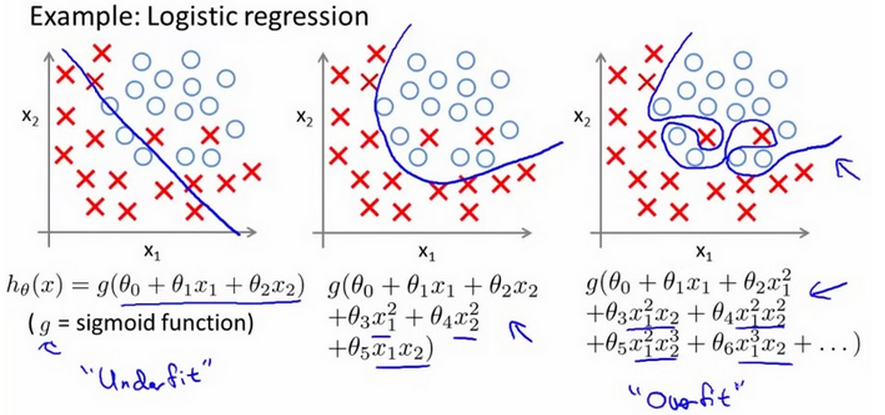


请一定要注意，求解似然函数一般是求解最大值，使用梯度上升算法，使得最优解不断增大。相对应的是损失函数，一般是似然函数的相反数，所以使用的是梯度下降法求解损失函数的最小值参数。

通过推导我们发现逻辑斯特回归的代价函数与线性回归形式上很像，不同之处在于模型假设不一样，线性回归是， 而逻辑回归在此基础上多了一层映射。

上述的似然函数，我们可以在目标函数中加一个负号变为损失函数或代价函数，从而原来的梯度上升的算法就变为了梯度下降算法求解参数的值。

##### （6）通过正则化解决拟合问题



过拟合的问题是机器学习中常见的问题，它是指模型在训练数据上表现良好，但是用于未知数据（测试数据）时性能不佳。如果一个模型出现了过拟合的问题，我们也说这个模型有高方差，这就有可能是因为使用了相关数据中过多的参数，从而使得模型变得复杂。同样，模型也可能出现欠拟合，这意味着模型过于简单，无法发现训练数据集中的隐含的模式，这也会使得模型应用于未知数据的时候表现不佳。

偏差-方差权衡bias和variance就是通过正则化调整模型的复杂度，我们通过正则化解决共线性（特征高度相似的）一个很用的办法。它可以过滤掉数据中的噪声。具体的做法就是引入额外的信息或偏差对极端的参数权重做出惩罚。

最常用的是L2正则化，有时也称为L2收缩或衰减，写作如下公式：

，为正则化系数

在逻辑斯特函数中，只需要在最大似然函数中加入正则化项，以降低系数带来的副作用：

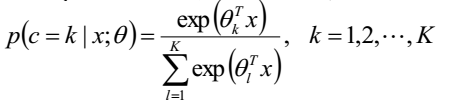


通过正则化系数，保持权值较小的时候，我们就可以控制模型与训练数据的拟合程度，加入值，可以增强正则化的强度。

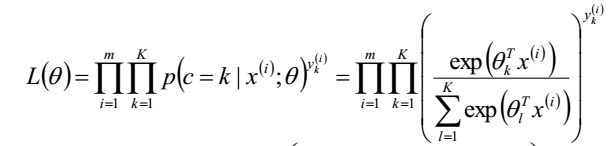
##### （7）Softmax多分类

多类别处理问题方法

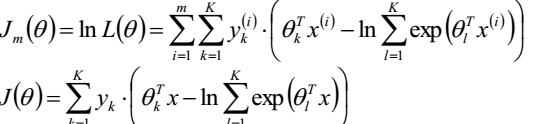
1.类别概率



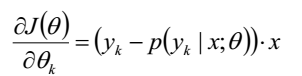
2.似然函数



3.对数似然



4.随机梯度



#### 2.2.6SparkMl逻辑回归问题实战

逻辑回归是预测分类应用的常用方法。广义线性模型的一个特例是预测结果的概率。在spark.ml逻辑回归中，可以使用二项逻辑回归来预测二元结果，或者可以使用多项逻辑回归来预测多类结果。使用该family 参数在这两种算法之间进行选择，或者保持不设置，Spark将推断出正确的变量。

通过将family参数设置为“多项式”，可以将多项逻辑回归用于二进制分类。它将产生两组系数和两个截距。

当在不具有常量非零列的数据集上截断LogisticRegressionModel时，Spark MLlib为常量非零列输出零系数。此行为与R glmnet相同，但与LIBSVM不同。

##### Binomial logistic regression

import org.apache.spark.ml.classification.LogisticRegression

// Load training data

val training = spark.read.format("libsvm").load("data/mllib/sample\_libsvm\_data.txt")

val lr = new LogisticRegression()

.setMaxIter(10)

.setRegParam(0.3)

.setElasticNetParam(0.8)

// Fit the model

val lrModel = lr.fit(training)

// Print the coefficients and intercept for logistic regression

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

// We can also use the multinomial family for binary classification

val mlr = new LogisticRegression()

.setMaxIter(10)

.setRegParam(0.3)

.setElasticNetParam(0.8)

.setFamily("multinomial")

val mlrModel = mlr.fit(training)

// Print the coefficients and intercepts for logistic regression with multinomial family

println(s"Multinomial coefficients: ${mlrModel.coefficientMatrix}")

println(s"Multinomial intercepts: ${mlrModel.interceptVector}")

spark.ml逻辑回归的实现还支持在训练集上提取模型的摘要summary信息。请注意，这是存储为预测和指标DataFrame在LogisticRegressionSummary被注释@transient，仅适用于驱动程序。

[LogisticRegressionTrainingSummary](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.ml.classification.LogisticRegressionTrainingSummary) 提供了摘要 [LogisticRegressionModel](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.ml.classification.LogisticRegressionModel)。在二进制分类的情况下，可以使用某些附加度量，例如ROC曲线。可以通过该binarySummary方法访问二分类信息-，[BinaryLogisticRegressionTrainingSummary](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.ml.classification.BinaryLogisticRegressionTrainingSummary) 。多分类可能会有所变化。

继续前面的例子：

import org.apache.spark.ml.classification.{BinaryLogisticRegressionSummary, LogisticRegression}

// Extract the summary from the returned LogisticRegressionModel instance trained in the earlier

// example

val trainingSummary = lrModel.summary

// Obtain the objective per iteration.

val objectiveHistory = trainingSummary.objectiveHistory

println("objectiveHistory:")

objectiveHistory.foreach(loss => println(loss))

// Obtain the metrics useful to judge performance on test data.

// We cast the summary to a BinaryLogisticRegressionSummary since the problem is a

// binary classification problem.

val binarySummary = trainingSummary.asInstanceOf[BinaryLogisticRegressionSummary]

// Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.

val roc = binarySummary.roc

roc.show()

println(s"areaUnderROC: ${binarySummary.areaUnderROC}")

// Set the model threshold to maximize F-Measure

val fMeasure = binarySummary.fMeasureByThreshold

val maxFMeasure = fMeasure.select(max("F-Measure")).head().getDouble(0)

val bestThreshold = fMeasure.where($"F-Measure" === maxFMeasure)

.select("threshold").head().getDouble(0)

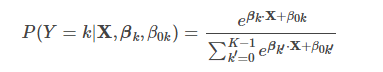
lrModel.setThreshold(bestThreshold)

##### Multinomial logistic regression

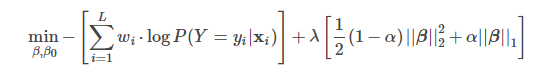
通过多项logistic（softmax）回归支持多类分类。在多项Logistic回归中，算法产生ķ个系数集合或维数k的矩阵ķ×J，ķ是类别标签类的个数，J是特征的数量。如果算法训练截距项部分，则长度为ķ 截距项就可以使用。

多项系数可用coefficientMatrix，截距可用interceptVector。coefficientsintercept不支持使用多项式训练的逻辑回归模型和方法。使用coefficientMatrix和interceptVector替代。

该结果类的条件概率：



算法使用多项式响应模型最小化加权负对数似然，使用弹性网惩罚来控制过度拟合。



代码：

import org.apache.spark.ml.classification.LogisticRegression

// Load training data

val training = spark

.read

.format("libsvm")

.load("data/mllib/sample\_multiclass\_classification\_data.txt")

val lr = new LogisticRegression()

.setMaxIter(10)

.setRegParam(0.3)

.setElasticNetParam(0.8)

// Fit the model

val lrModel = lr.fit(training)

// Print the coefficients and intercept for multinomial logistic regression

println(s"Coefficients: \n${lrModel.coefficientMatrix}")

println(s"Intercepts: ${lrModel.interceptVector}")

#### 2.2.7SparkMllib逻辑回归API实战

[Logistic回归](http://en.wikipedia.org/wiki/Logistic_regression)广泛用于预测二分类问题。其中由逻辑损失给出的公式中的损失函数：



对于二进制分类问题，算法输出二元逻辑回归模型。给定一个新的数据点，用z表示特征，使用logistic function预测 ，其中



这里z=wTx， 如果f(wTx)>0.5, 结果为正,否则结果为负

我们实现了两种算法来解决逻辑回归：小批量梯度下降和L-BFGS。这里建议L-BFGS超过小批量梯度下降，以实现更快的收敛。

代码部分：

import org.apache.spark.mllib.classification.{LogisticRegressionModel, LogisticRegressionWithLBFGS}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.util.MLUtils

// 1-Load training data in LIBSVM format.

val data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample\_libsvm\_data.txt")

//2- Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

//3- Run training algorithm to build the model

val model = new LogisticRegressionWithLBFGS()

.setNumClasses(10)

.run(training)

// 4-Compute raw scores on the test set.

val predictionAndLabels = test.map { case LabeledPoint(label, features) =>

val prediction = model.predict(features)

(prediction, label)

}

// 5-Get evaluation metrics.

val metrics = new MulticlassMetrics(predictionAndLabels)

val accuracy = metrics.accuracy

println(s"Accuracy = $accuracy")

// 6-Save and load model

model.save(sc, "target/tmp/scalaLogisticRegressionWithLBFGSModel")

val sameModel = LogisticRegressionModel.load(sc,

"target/tmp/scalaLogisticRegressionWithLBFGSModel")

#### 2.2.8Ctr点击率预估场景问题实战

参考Ctr预估实战部分课件

## SparkMllib回归任务实战详解

### 3.1SparkMllib回归算法分类

回归模型属于监督式学习，每个个体都有一个与之相关联的实数标签，并且我们希望在给出用于表示这些实体的数值特征后，所预测出的标签值可以尽可能接近实际值。

回归算法是试图采用对误差的衡量来探索变量之间的关系的一类算法。回归算法是统计机器学习的利器。在机器学习领域，人们说起回归，有时候是指一类问题，有时候是指一类算法，这一点常常会使初学者有所困惑。常见的回归算法包括：普通最小二乘法（OLS）（Ordinary Least Square），它使用损失函数是平方损失函数（1/2 (w^T x-y)2），简单的预测就是y=wT x，标准的最小二乘回归不使用正则化，这就意味着数据中异常数据点非常敏感，因此，在实际应用中经常使用一定程度的正则化（目的避免过拟合、提供泛化能力）。

SparkMllib目前支持**线性回归，广义线性回归，决策树回归，随机森林回归，梯度提升书回归，生存回归Survial regression，Isotonic regression保序回归。**

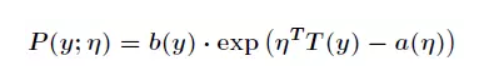
### 3.2SparkMllib-GeneralizedLinearAlgorithm原理及实战

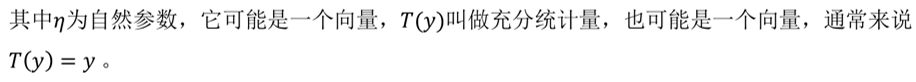
**广义线性回归：**

回归方式比较常用的有线性回归和logistic回归.基本的形式都是先设定h\_θ (x)，然后求最最大似然估计L(θ),然后求出l(θ)=logL(θ),然后用梯度上升法或其它方法求出θ，二种回归如此相似的原因就是在于它们都是广义线性模型里的一员。所以为了有个总体上的把握，从广义线性回归说起。

**指数分布族：**

若概率分布满足下式，我们就称之属于指数分布族：





当固定T时，分布属于指数家族中的哪种分布就由a和b两个函数决定。

**构建广义线性回归步骤：**

考虑一个分类或回归问题，我们就是想预测某个随机变量y，y 是某些特征(feature)x的函数。为了推导广义线性模式，我们必须做出如下三个假设

（1）p(y|x;θ) 服从指数族分布

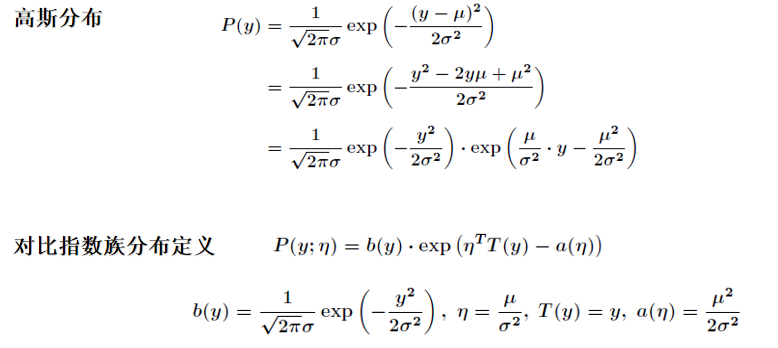
（2）给了x, 我们的目的是为了预测T(y)的在条件x下的期望。一般情况T(y)=y, 这就意味着我们希望预测h(x)=E[y|x]

（3）参数η和输入x 是线性相关的：η=θTx

上述讲到的线性回归中的最小二乘法，逻辑回归，softmax回归其实都是属于广义线性回归的一种。

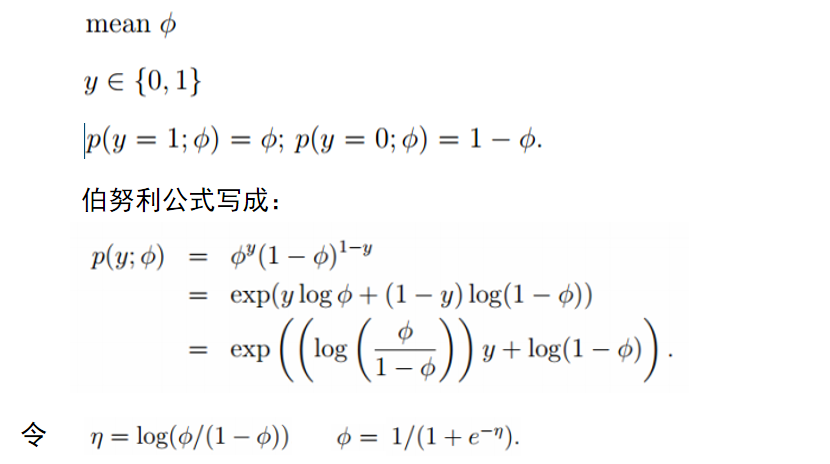
**接下来分析高斯分布和伯努利分布如何变化到指数分布族分布：**

如：指数分步族到高斯分布变化：



如：指数分布族-伯努利分布

**通过下面的证明说明logistic函数或sigmod函数输入数据需要满足伯努利分布(面试常考)**



Spark中定义的广义线性回归：

与线性回归相比，输出被假设为服从高斯分布，[**广义线性模型**]（GLM）是线性模型的规范，其中响应变量Yi遵循[指数族分布]的一些分布。Spark的GeneralizedLinearRegression界面允许灵活的GLM规范，可用于各种类型的预测问题，包括线性回归，泊松回归，逻辑回归等。 目前在spark.ml中，仅支持指数族分布的一部分

以下代码使用高斯分布和数据训练GLM模型并提取模型汇总统计信息

import org.apache.spark.ml.regression.GeneralizedLinearRegression

//1- Load training data

val dataset = spark.read.format("libsvm")

.load("data/mllib/sample\_linear\_regression\_data.txt")

//2-准备算法

val glr = new GeneralizedLinearRegression()

.setFamily("gaussian")

.setLink("identity")

.setMaxIter(10)

.setRegParam(0.3)

// 3-Fit the model

val model = glr.fit(dataset)

// 4-Print the coefficients and intercept for generalized linear regression model

println(s"Coefficients: ${model.coefficients}")

println(s"Intercept: ${model.intercept}")

// 5-Summarize the model over the training set and print out some metrics

val summary = model.summary

println(s"Coefficient Standard Errors: ${summary.coefficientStandardErrors.mkString(",")}")

println(s"T Values: ${summary.tValues.mkString(",")}")

println(s"P Values: ${summary.pValues.mkString(",")}")

println(s"Dispersion: ${summary.dispersion}")

println(s"Null Deviance: ${summary.nullDeviance}")

println(s"Residual Degree Of Freedom Null: ${summary.residualDegreeOfFreedomNull}")

println(s"Deviance: ${summary.deviance}")

println(s"Residual Degree Of Freedom: ${summary.residualDegreeOfFreedom}")

println(s"AIC: ${summary.aic}")

println("Deviance Residuals: ")

summary.residuals().show()

### 3.3SparkMllib-DecisionTreeRegression原理及实战

决策树是一种流行的分类和回归方法。下面以LibSVM格式加载数据集，将其拆分为训练和测试集，在第一个数据集上训练，然后对所保留的测试集进行评估。我们使用特征变换器来对分类特征进行索引，将元数据添加到决策树算法可以识别DataFrame中。

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.evaluation.RegressionEvaluator

import org.apache.spark.ml.feature.VectorIndexer

import org.apache.spark.ml.regression.DecisionTreeRegressionModel

import org.apache.spark.ml.regression.DecisionTreeRegressor

//1- Load the data stored in LIBSVM format as a DataFrame.

val data = spark.read.format("libsvm").load("data/mllib/sample\_libsvm\_data.txt")

//2- Automatically identify categorical features, and index them.

// Here, we treat features with > 4 distinct values as continuous.

val featureIndexer = new VectorIndexer()

.setInputCol("features")

.setOutputCol("indexedFeatures")

.setMaxCategories(4)

.fit(data)

// 3-Split the data into training and test sets (30% held out for testing).

val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))

//4- Train a DecisionTree model.

val dt = new DecisionTreeRegressor()

.setLabelCol("label")

.setFeaturesCol("indexedFeatures")

// 5-Chain indexer and tree in a Pipeline.

val pipeline = new Pipeline()

.setStages(Array(featureIndexer, dt))

//6- Train model. This also runs the indexer.

val model = pipeline.fit(trainingData)

// 7-Make predictions.

val predictions = model.transform(testData)

// 8-Select example rows to display.

predictions.select("prediction", "label", "features").show(5)

// 9-Select (prediction, true label) and compute test error.

val evaluator = new RegressionEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println(s"Root Mean Squared Error (RMSE) on test data = $rmse")

//10-保存模型

val treeModel = model.stages(1).asInstanceOf[DecisionTreeRegressionModel]

println(s"Learned regression tree model:\n ${treeModel.toDebugString}")

### 3.4SparkMllib-RandomforestRegression实战

随机森林是一类受欢迎的分类和回归方法。接下来以LibSVM格式加载数据集，将其拆分为训练和测试集，在第一个数据集上训练，然后对所保留的测试集进行评估。我们使用特征变换器对分类特征进行索引，将元数据添加到基于树的算法可以识别的DataFrame中。我们使用特征变换器对分类特征进行索引，将元数据添加到基于树的算法可以识别的

DataFrame中。

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.evaluation.RegressionEvaluator

import org.apache.spark.ml.feature.VectorIndexer

import org.apache.spark.ml.regression.{RandomForestRegressionModel, RandomForestRegressor}

// 1-Load and parse the data file, converting it to a DataFrame.

val data = spark.read.format("libsvm").load("data/mllib/sample\_libsvm\_data.txt")

//2- Automatically identify categorical features, and index them.

// Set maxCategories so features with > 4 distinct values are treated as continuous.

val featureIndexer = new VectorIndexer()

.setInputCol("features")

.setOutputCol("indexedFeatures")

.setMaxCategories(4)

.fit(data)

// 3-Split the data into training and test sets (30% held out for testing).

val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))

// 4-Train a RandomForest model.

val rf = new RandomForestRegressor()

.setLabelCol("label")

.setFeaturesCol("indexedFeatures")

// 5-Chain indexer and forest in a Pipeline.

val pipeline = new Pipeline()

.setStages(Array(featureIndexer, rf))

// 6-Train model. This also runs the indexer.

val model = pipeline.fit(trainingData)

// 7-Make predictions.

val predictions = model.transform(testData)

// 8-Select example rows to display.

predictions.select("prediction", "label", "features").show(5)

// 9-Select (prediction, true label) and compute test error.

val evaluator = new RegressionEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println(s"Root Mean Squared Error (RMSE) on test data = $rmse")

//10-save model

val rfModel = model.stages(1).asInstanceOf[RandomForestRegressionModel]

println(s"Learned regression forest model:\n ${rfModel.toDebugString}")

### 3.5SparkMllib-GBRT实战

案例：GBTR实现。注意：对于以下数据集，GBTRegressor实际上只需要1次迭代，但这一般不会仅有1次。

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.evaluation.RegressionEvaluator

import org.apache.spark.ml.feature.VectorIndexer

import org.apache.spark.ml.regression.{GBTRegressionModel, GBTRegressor}

//1- Load and parse the data file, converting it to a DataFrame.

val data = spark.read.format("libsvm").load("data/mllib/sample\_libsvm\_data.txt")

//2- Automatically identify categorical features, and index them.

// Set maxCategories so features with > 4 distinct values are treated as continuous.

val featureIndexer = new VectorIndexer()

.setInputCol("features")

.setOutputCol("indexedFeatures")

.setMaxCategories(4)

.fit(data)

// 3-Split the data into training and test sets (30% held out for testing).

val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))

//4- Train a GBT model.

val gbt = new GBTRegressor()

.setLabelCol("label")

.setFeaturesCol("indexedFeatures")

.setMaxIter(10)

// 5-Chain indexer and GBT in a Pipeline.

val pipeline = new Pipeline()

.setStages(Array(featureIndexer, gbt))

//6- Train model. This also runs the indexer.

val model = pipeline.fit(trainingData)

// 7-Make predictions.

val predictions = model.transform(testData)

// 8-Select example rows to display.

predictions.select("prediction", "label", "features").show(5)

// 8-Select (prediction, true label) and compute test error.

val evaluator = new RegressionEvaluator()

.setLabelCol("label")

.setPredictionCol("prediction")

.setMetricName("rmse")

val rmse = evaluator.evaluate(predictions)

println(s"Root Mean Squared Error (RMSE) on test data = $rmse")

val gbtModel = model.stages(1).asInstanceOf[GBTRegressionModel]

println(s"Learned regression GBT model:\n ${gbtModel.toDebugString}")

### 3.6SparkMllib-SurvivalRegression原理及实战

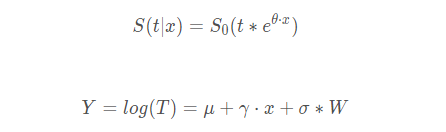
在spark.ml中，我们实现加速失效时间模型（[Acceleratedfailure time](https://en.wikipedia.org/wiki/Accelerated_failure_time_model" \t "https://blog.csdn.net/liulingyuan6/article/details/_blank)），对于截尾数据它是一个**参数化生存回归的模型**。它描述了一个有对数生存时间的模型，所以它也常被称为**生存分析的对数线性模型**。与比例危险模型不同，因AFT模型中每个实例对目标函数的贡献是独立的，其更容易并行化。

加速失效（AFT）模型

假设 T 为失效时间，x 为协变量，加速失效（accelerate failure time）模型的假设是，**一个人的生存时间等于人群基准生存时间 \* 这个人的加速因子**，其数学形式如下：



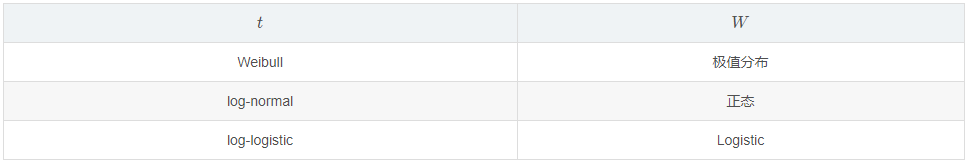
常见的形式还有(**生存分析的对数线性模型**)：



参数符号说明：

* γ=−θ
* θ 称作加速系数，而 γ 常常被称为回归系数

其中人群的基准生存时间t 服从某个概率分布，W 也是服从某个概率分布的随机变量。常见分布如下：



从上面的数学形式可以看出，某个人的生存时间受基准生存时间和加速因子的影响的，而基准生存时间为研究中所有人共有，加速因子则与该人与研究目标相关的协变量相关。

所以，很显然，该模型需要估计的参数为与基准生存时间分布相关的参数μ,σ，以及和协变量相关的系数θ。

参数估计与推断由极大似然估计得到其结果。

SparkMllib支持的参数：

（1）censorCol:

类型：字符串型。

含义：检查器列名。

（2）featuresCol:

类型：字符串型。

含义：特征列名。

（3）fitIntercept:

类型：布尔型。

含义：是否训练拦截对象。

（4）labelCol:

类型：字符串型。

含义：标签列名。

（5）maxIter:

类型：整数型。

含义：迭代次数（>=0）。

（6）quantileProbabilities:

类型：双精度数组型。

含义：分位数概率数组。

（7）quantilesCol:

类型：字符串型。

含义：分位数列名。

（8）stepSize:

类型：双精度型。

含义：每次迭代优化步长。

（9）tol:

类型：双精度型。

含义：迭代算法的收敛性。

代码如下：

import org.apache.spark.ml.linalg.Vectors

import org.apache.spark.ml.regression.AFTSurvivalRegression

val training = spark.createDataFrame(Seq(

(1.218, 1.0, Vectors.dense(1.560, -0.605)),

(2.949, 0.0, Vectors.dense(0.346, 2.158)),

(3.627, 0.0, Vectors.dense(1.380, 0.231)),

(0.273, 1.0, Vectors.dense(0.520, 1.151)),

(4.199, 0.0, Vectors.dense(0.795, -0.226))

)).toDF("label", "censor", "features")

val quantileProbabilities = Array(0.3, 0.6)

val aft = new AFTSurvivalRegression()

.setQuantileProbabilities(quantileProbabilities)

.setQuantilesCol("quantiles")

val model = aft.fit(training)

// Print the coefficients, intercept and scale parameter for AFT survival regression

println(s"Coefficients: ${model.coefficients}")

println(s"Intercept: ${model.intercept}")

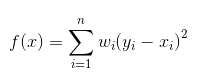
println(s"Scale: ${model.scale}")

model.transform(training).show(false)

### 3.7SparkMllib-IsotonicRegression原理及实战

算法介绍：

保序回归是回归算法的一种。保序回归给定一个有限的实数集合IMG_256 代表观察到的响应，以及IMG_257 代表未知的响应值，训练一个模型来最小化下列方程：



其中 ，IMG_260 为权重是正值。其结果方程称为保序回归，而且其解是唯一的。它可以被视为有顺序约束下的最小二乘法问题。实际上保序回归在拟合原始数据点时是一个单调函数。我们实现池旁者算法，它使用并行保序回归。训练数据是DataFrame格式，包含标签、特征值以及权重三列。另外保序算法还有一个参数名为isotonic，其默认值为真，它指定保序回归为保序（单调递增）或者反序（单调递减）。

训练返回一个保序回归模型，可以被用于来预测已知或者未知特征值的标签。保序回归的结果是分段线性函数，预测规则如下：

1.如果预测输入与训练中的特征值完全匹配，则返回相应标签。如果一个特征值对应多个预测标签值，则返回其中一个，具体是哪一个未指定。

2.如果预测输入比训练中的特征值都高（或者都低），则相应返回最高特征值或者最低特征值对应标签。如果一个特征值对应多个预测标签值，则相应返回最高值或者最低值。

3.如果预测输入落入两个特征值之间，则预测将会是一个分段线性函数，其值由两个最近的特征值的预测值计算得到。如果一个特征值对应多个预测标签值，则使用上述两种情况中的处理方式解决。

SparkMllib中的保序回归的参数：

（1）featuresIndex:

类型：整数型。

含义：当特征列维向量时提供索引值，否则不进行处理。

（2）featuresCol:

类型：字符串型。

含义：特征列名。

（3）isotonic:

类型：布尔型。

含义：输出序列为保序/增序（真）或者反序/降序（假）。

（4）labelCol:

类型：字符串型。

含义：标签列名。

（5）predictionCol:

类型：字符串型。

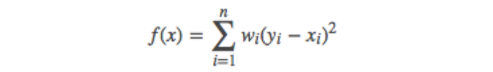
含义：预测结果列名。

（6）weightCol:

类型：字符串型。

含义：列权重。

**[保序回归解析]**属于回归算法族。正则等式回归是一个问题，其中给出了一个有限的实数集Y = y1，y2，...，表示观察到的响应，X = x1，x2，...，xn 要拟合的未知响应值，找到最小化的函数：



对于满足x1≤x2≤...≤xn的完整数据，其中wi是正权重。所得到的函数称为等渗回归，它是唯一的。它可以被视为变量限制下的最小二乘问题。基本等渗回归是最接近原始数据点的单调函数。

我们实施一个相邻不同于算法的池，它使用一种方法来平行等渗回归。训练输入是一个DataFrame，它包含三列标签，特征和重量。另外，IsotonicRegression算法有一个可选参数，称为等渗偏差默认为true。这个参数指定等渗回归是等渗的（单调递增）还是反（单调递减）。

训练返回一个可以用于预测已知和未知特征的标签的等渗回归模型。等渗回归的结果被视为分段线性函数。

因此，预测规则是：

* 如果预测输入与训练特征完全匹配，则返回相关预测。 如果有多个具有相同特征的预测，则返回其中一个。 哪一个是未定义的（与java.util.Arrays.binarySearch相同）。
* 如果预测输入低于或高于所有训练特征，则返回具有最低或最高特征的预测。 如果存在具有相同特征的多个预测，则分别返回最低或最高。
* 如果预测输入落在两个训练特征之间，则预测被视为分段线性函数，并且根据两个最接近的特征的预测来计算内插值。 如果有多个具有相同特征的值，则使用与前一点相同的规则

import org.apache.spark.ml.regression.IsotonicRegression

//1- Loads data.

val dataset = spark.read.format("libsvm")

.load("data/mllib/sample\_isotonic\_regression\_libsvm\_data.txt")

//2- Trains an isotonic regression model.

val ir = new IsotonicRegression()

val model = ir.fit(dataset)

println(s"Boundaries in increasing order: ${model.boundaries}\n")

println(s"Predictions associated with the boundaries: ${model.predictions}\n")

//3- Makes predictions.

model.transform(dataset).show()