|  |
| --- |
|  |
| 机器学习基础 |
|  |

**\*\*\*\*\*\*\*\*\*\*\*\***

2013-10-16

作者：钟超

机器学习基础

目录

[一 机器学习概述 4](#_Toc376721495)

[1.1 统计学习 4](#_Toc376721496)

[1.2 监督学习 4](#_Toc376721497)

[1.3 模型评估与选择 4](#_Toc376721498)

[1.4 模型的泛化能力 5](#_Toc376721499)

[二 预测算法 5](#_Toc376721500)

[1 一元线性回归 5](#_Toc376721501)

[1.1 为什么用回归 5](#_Toc376721502)

[1.2 一元线性回归模型 5](#_Toc376721503)

[2 最优化方法-梯度下降法 7](#_Toc376721504)

[3 基函数 10](#_Toc376721505)

[3.1 多项式回归 10](#_Toc376721506)

[3.2 回归模型中的基函数 13](#_Toc376721507)

[4 欠拟合与过拟合 13](#_Toc376721508)

[4.1 欠拟合 13](#_Toc376721509)

[4.2 过拟合 13](#_Toc376721510)

[5 多元线性回归 15](#_Toc376721511)

[6 应用实例- 18](#_Toc376721512)

[三 分类算法 18](#_Toc376721513)

[1 线性分类器-感知器 18](#_Toc376721514)

[1.1 感知器 18](#_Toc376721515)

[1.2 感知器的学习策略 18](#_Toc376721516)

[1.3 优化损失函数 19](#_Toc376721517)

[1.4 代码实现 19](#_Toc376721518)

[2 线性分类器-逻辑回归 20](#_Toc376721519)

[2.1 逻辑回归分布 21](#_Toc376721520)

[2.2 二项逻辑回归 21](#_Toc376721521)

[2.3 参数估计 21](#_Toc376721522)

[2.4 基函数 22](#_Toc376721523)

[2.5 过拟合（正则化） 22](#_Toc376721524)

[2.6 参数的矩阵表示 22](#_Toc376721525)

[2.7 代码实现 22](#_Toc376721526)

[3 贝叶斯分类器 24](#_Toc376721527)

[3.1 贝叶斯公式 24](#_Toc376721528)

[3.2 高斯贝叶斯分类器 24](#_Toc376721529)

[3.2.1 理论简介 24](#_Toc376721530)

[3.2.2 代码实现 25](#_Toc376721531)

[3.3 多项式贝叶斯分类器 25](#_Toc376721532)

[3.3.1、构造数据集信息 26](#_Toc376721533)

[3.3.2、计算特征（单词）概率 26](#_Toc376721534)

[3.3.3、计算整篇文档的频率 27](#_Toc376721535)

[3.3.4、贝叶斯公式 27](#_Toc376721536)

[3.3.5、选择分类 28](#_Toc376721537)

[3.3.6、费舍尔方法 28](#_Toc376721538)

[3.3.7、增量式训练 28](#_Toc376721539)

[4 总结 29](#_Toc376721540)

[5 应用实例-主动客服 29](#_Toc376721541)

[四 聚类算法 29](#_Toc376721542)

[1 Kmeans 29](#_Toc376721543)

[2 谱聚类 32](#_Toc376721544)

[3 应用实例-网格化配送 34](#_Toc376721545)

[五 降维算法 34](#_Toc376721546)

[1 主成分分析（PCA） 34](#_Toc376721547)

[1.1 主成分应用 34](#_Toc376721548)

[1.2 一个例子 35](#_Toc376721549)

[1.2.1 求主成分和主成分得分 35](#_Toc376721550)

[1.2.2 确定分析精度 37](#_Toc376721551)

[1.2.3 分析结果 37](#_Toc376721552)

[1.2.4 程序解析 38](#_Toc376721553)

[2 隐性语意分析（LSA） 38](#_Toc376721554)

[2.1 基于LSA的文本摘要算法 38](#_Toc376721555)

[2.2 文本降维 40](#_Toc376721556)

[3 应用实例-文本聚类 41](#_Toc376721557)

[六 模型选择 41](#_Toc376721558)

[1 模型选择方法 41](#_Toc376721559)

[2 交叉验证 41](#_Toc376721560)

[**2.1 计算交叉验证指标** 42](#_Toc376721561)

[2.2 数据集分割方法 42](#_Toc376721562)

[**2.2.1 K折法** 42](#_Toc376721563)

[**2.2.2 留一验证法（LOO）** 42](#_Toc376721564)

[**2.2.3 留P个样本验证（LPO）** 42](#_Toc376721565)

[3 模型性能的评价准则 42](#_Toc376721566)

[3.1 混淆矩阵 42](#_Toc376721567)

[3.2 准确率、召回率、F-得分 43](#_Toc376721568)

[七 推荐算法 44](#_Toc376721569)

[7.1、推荐系统概述 44](#_Toc376721570)

[7.2、推荐系统类型 44](#_Toc376721571)

[7.3、推荐系统的组成 44](#_Toc376721572)

[7.4、基于协同过滤的推荐 45](#_Toc376721573)

[7.4.1 什么是协同过滤 45](#_Toc376721574)

[7.4.2 协同过滤的核心 45](#_Toc376721575)

[7.4.3 协同过滤算法 45](#_Toc376721576)

[4.3.1 基于用户的协同过滤推荐 45](#_Toc376721577)

[4.3.2 基于项目的协同过滤推荐算法 46](#_Toc376721578)

[7.5 推荐算法评价 47](#_Toc376721579)

[7.5.1 训练数据和得分 47](#_Toc376721580)

[7.5.2 准确率和召回率 48](#_Toc376721581)

[八 技术前沿 50](#_Toc376721582)

[1 深度学习 50](#_Toc376721583)

[2 流形学习 50](#_Toc376721584)

[3 知识图谱 50](#_Toc376721585)

[4 推荐阅读 50](#_Toc376721586)

# 一 机器学习概述

## 1.1 统计学习

**1 学习方法**

监督学习，非监督学习，半监督学习

**2 统计学习三要素**

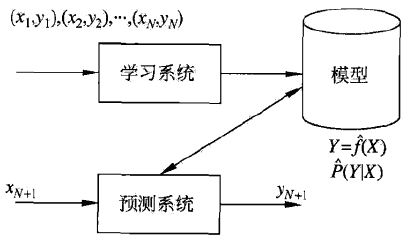
模型，策略，算法

## 1.2 监督学习

**1 基本概念**

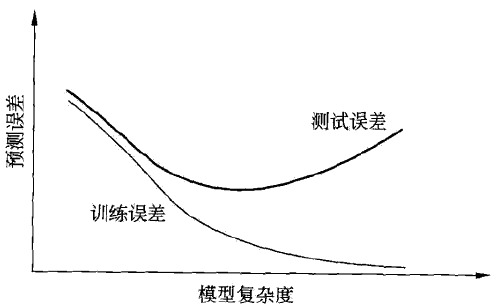
输入、输出空间，特征空间，假设空间

**2 监督学习过程**

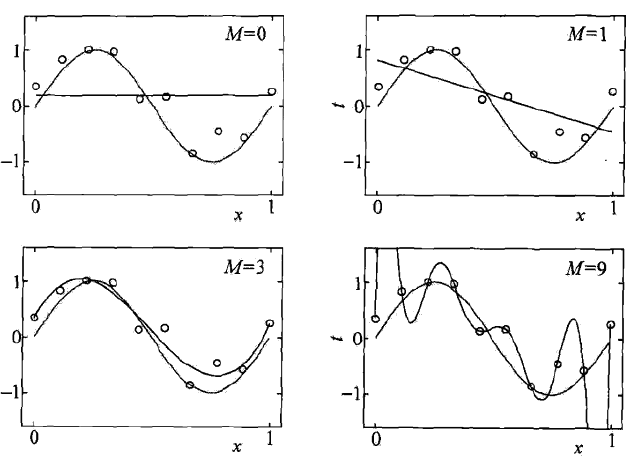


## 1.3 模型评估与选择

**1 训练误差与测试误差**



**2 过拟合与正则化**



**3 交叉验证**

**4 评价指标**

准确率和召回率

### 1.4 模型的泛化能力

即预测模型对样本的预测能力

# 二 预测算法

## 1 一元线性回归

### 1.1 为什么用回归

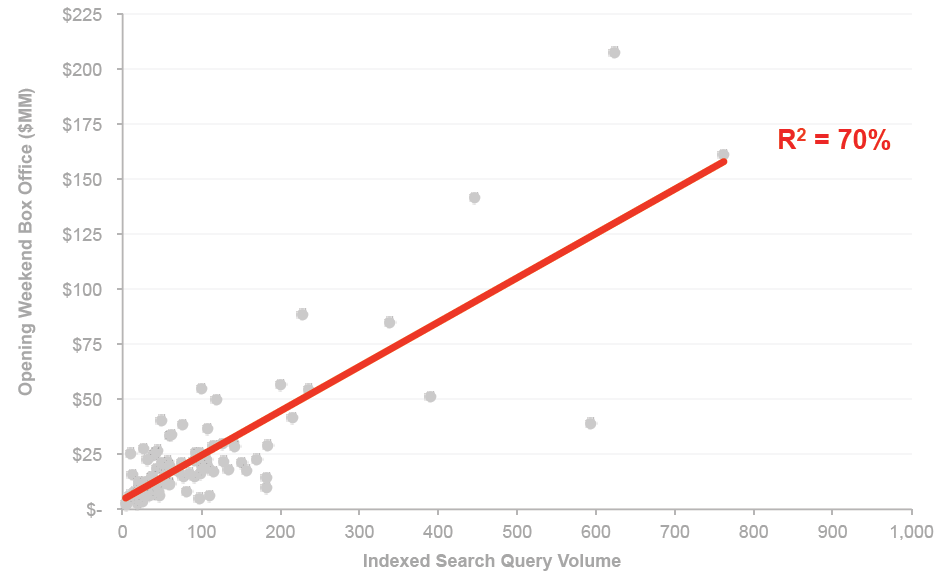


图1.1.1 Google的票房与搜索量的关系

图1.1显示的是Google发布的电影的搜索量与票房的关系。如何用历史的信息预测票房就是（线性）回归问题。

### 1.2 一元线性回归模型

**1 数学描述**

图1.1.1中的横、纵轴分别用用表示，。假设图1.1中使用的一元线性模型的形式为：

（1.2-1）

显然只要求出线性模型就可以确定了。为了求解系数需要构造一个目标函数（损失函数），如下

（1.2-2）

只要最小化式（1.2-2），就可以求出系数。这种做法非常直观，就是要使预测的结果和真值之间的差最小。

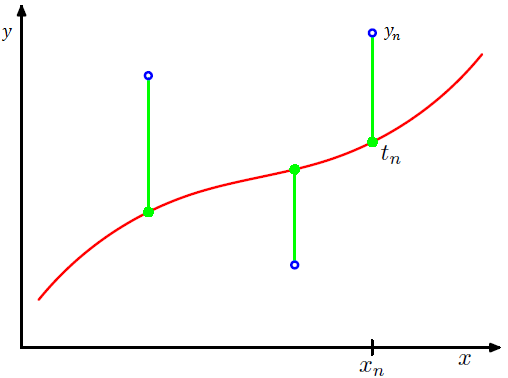


图1.2.1 函数的几何解释

是一个非负值，最小值为0，它的几何解释如图1.2.1，就是要使的距离平方和最小，回归函数要穿过真实数据。

**2 矩阵表示**

对于N各数据点，式（1.2-1）有N个等式，并用线性代数表示为

（1.2-3）

其中

，，

此时

（1.2-4）

其中

所以式（1.2.2）又可以表示为

（1.2-5）

**【因为，对于，】**

**3 目标函数最小化**

在高等数学中，使函数一阶导数为0，且二阶导数要大于0的点为函数的最小值点。式（1.2-5）所表示的是二次函数且开口向上，只要求一阶导数为0即可

（1.2-6）

【矩阵求导：；；，

，设第一项中，第二项中，第三项中，所以，

所以，

**1 详细推导**

**2 详细推导**

是维的，第个元素为

】

所以线性模型所对应的最优参数表示如下，也称为正则解或者闭形式解

（1.2-7）

一个简单的例子，代码见文件夹1\_regression。

第一步：用synthic\_data.py中的linearSamples方法生成数据

**import numpy as np**

**import random**

**def linearSamples(n = 20):**

**a = 0.5**

**b = 1.0**

**r = [i + 2.0\*random.random() for i in xrange(n)]**

**return [range(0, len(r)), r]**

第二步：用linear\_regression.py中的lR方法，完成式（1.2-7），最终的结果为的值

**def lR(x, y):**

**x = np.matrix(x)**

**if x.shape[0] == 1:**

**x = x.transpose()**

**y = np.matrix(y)**

**if y.shape[0] == 1:**

**y = y.transpose()**

**one = np.ones((x.shape[0], 1))**

**x = np.hstack([one, x])**

**w = inv((x.transpose()).dot(x)).dot(np.transpose(x)).dot(y)**

**return w**

第三步：将第二步中计算的和第一步中生成的数据，传递给plotLM方法，画出的数据点和回归直线如图1.2.2

**def plotLM(w, x,y):**

**xx = [i for i in np.arange(0.0,20.0,0.5)]**

**yy = [w[0,0] + w[1,0] \* i for i in xx]**

**fig = plt.figure()**

**ax = fig.add\_subplot(111)**

**ax.plot(x,y, '.')**

**ax.plot(xx,yy)**

**s = 'y = %s + %s \* x' %(str(w[0,0])[0:7], str(w[1, 0])[0:7])**

**ax.annotate(s, xy=(12.5, 13.3), xycoords='data',**

**xytext=(-180, 30), textcoords='offset points',**

**bbox=dict(boxstyle="round", fc="0.8"),**

**arrowprops=dict(arrowstyle="->",**

**connectionstyle="angle,angleA=0,angleB=90,rad=10"))**

**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.legend(('training sampes','regression line'))**

**plt.show()**

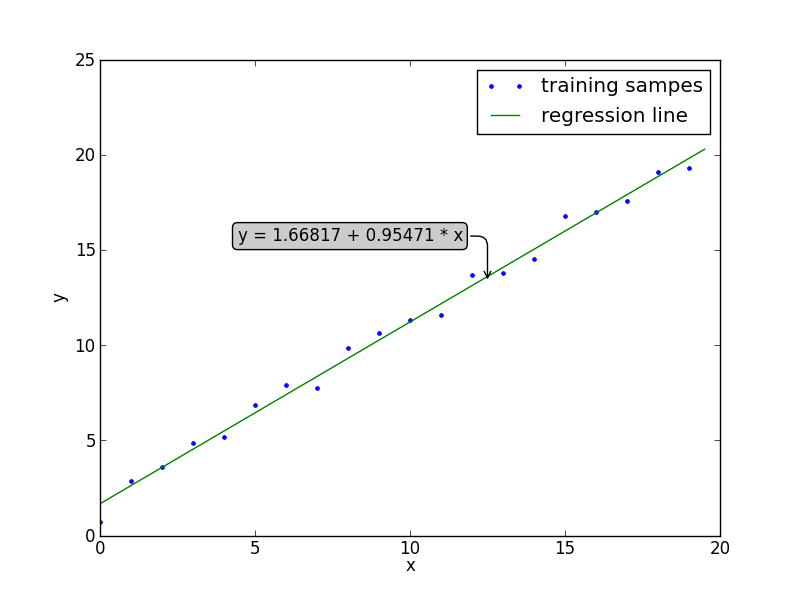


图1.2.2 线性回归的例子

在图1.2.2中散点代表的是训练数据，训练数据是由程序随机生成，没有实际意义，直线是回归直线，并标出了直线方程，在运行程序时直线结果可能与图中的结果稍有不同，因为训练数据是随机生成的缘故。

## 2 最优化方法-梯度下降法

在第一节的第3部分介绍了，将损失函数表示成矩阵形式，然后求导方法，求出最优的，这种方法对线性问题可以求出最优解，称为闭形式解或者解析解。本节介绍的梯度下降法是数值最优化方法，普适性更强，对于非线性问题依然可以求解。

梯度下降法是最常用、也是最容易理解的最优化方法。学会了梯度下降法，其它基于梯度的改进方法：共轭梯度法、牛顿法、拟牛顿法等，就比较容易理解。

**1 盲人是如何下山的**

第一步：左踩一脚，右踩一脚，如果发现这两脚在在高度上没有差别，此时他所面对的应该是山顶或者山脚，反之盲人面对的应该是山脊。（计算偏导数）

第二步：上踩一脚，下踩一脚，脚低的那个方向就对着山脚。（计算偏导数）

第三步：四个脚中，高度最低的那个方向就是山脚，从当前位置向下夸一小步，向着山脚进发。（确定步长，学习率）

重复第一、二、三步，直到山脚。

**2 梯度下降法**

梯度法就和盲人下山类似，就两个步骤：**首先确定下山方，然后再确定的方向上按照一定的步长下山**。

下面介绍最优化问题。

单目标、无约束、多维最优化问题的数学描述：

其中，。

梯度下降法算法流程如下：

1）给定初值，精度0，并令。

2）计算梯度下降方向（搜索方向），表示在处的梯度。

【所表示的是数值梯度，求法如下：

其中，】

3）若，则停止计算，否则从出发，沿着一维搜索，即求，使的，此处的一维搜索可以用黄金分割法或者二次差值法等。

4）令，，转2）。

**3 基于梯度下降法的线性回归**

下面用用梯度下降法，优化目标函数：式（1.2.2），并给出相应的代码解释

第一步：依然使用linearSamples生成数据，代码见前文

第二步：完成目标函数的定义，即式（1.2-2）

**def obj(x, y, w):**

**t = x.dot(w) - y**

**t = np.multiply(t, t)**

**sum\_ = 0.5 \* np.sum(t)**

**return sum\_**

第三步：完成数值梯度的定义，按照梯度下降法中的第2）点介绍，完成代码编写

**def gradient(fun, x, y, w, delta = 1e-6, \*args):**

**l = len(w)**

**g = []**

**for i in range(0, l):**

**delta\_w = deepcopy(w)**

**delta\_w[i] = delta\_w[i] + delta**

**g.append(-(obj(x, y, delta\_w) - obj(x, y, w))/delta)**

**return g**

第四步：gdLR方法将实现，梯度下降法中介绍的流程1），2），4），忽略了第三步，其中的学习率由手动调整。计算结束后返回最优的。

**def gdLR(fun, x, y, step = 0.0007,tol = 1e-6):**

**#preprocess the data**

**x = np.matrix(x)**

**if x.shape[0] == 1:**

**x = x.transpose()**

**y = np.matrix(y)**

**if y.shape[0] == 1:**

**y = y.transpose()**

**one = np.ones((x.shape[0], 1))**

**x = np.hstack([one, x])**

**w = [0.0, 0.0]**

**w = np.matrix(w)**

**if w.shape[0] == 1:**

**w = w.transpose()**

**l = len(w)**

**k = 1**

**while(True):**

**step1 = step / k**

**#1)compute negative gradient**

**g = gradient(fun, x, y, w)**

**err = linalg.norm(g)**

**print err**

**if err < tol or k > 200:**

**break**

**#2)updata the parameters**

**w = [w[i,0] + step \* g[i] for i in range(0, l)]**

**w = np.matrix(w).transpose()**

**k = k + 1**

**return w**

第五步：将闭形式的和梯度下降法的，以及数据x,y传递给方法plotGdLM画出对比图，见图2.1。

**def plotGdLM(cf\_w,gd\_w, x,y):**

**xx = [i for i in np.arange(0.0,20.0,0.5)]**

**cf\_yy = [cf\_w[0,0] + cf\_w[1,0] \* i for i in xx]**

**gd\_yy = [gd\_w[0,0] + gd\_w[1,0] \* i for i in xx]**

**fig = plt.figure()**

**ax = fig.add\_subplot(111)**

**ax.plot(x,y, '.')**

**ax.plot(xx,cf\_yy,color = 'g', linewidth=3)**

**s = 'y = %s + %s \* x' %(str(cf\_w[0,0])[0:7], str(cf\_w[1, 0])[0:7])**

**ax.annotate(s, xy=(12.5, 13.3), xycoords='data',**

**xytext=(-180, 30), textcoords='offset points',**

**bbox=dict(boxstyle="round", fc='g', ec='g'),**

**arrowprops=dict(arrowstyle="->",fc='g', ec='g',**

**connectionstyle="angle,angleA=0,angleB=90,rad=10"))**

**ax.plot(xx,gd\_yy, color = 'r', linewidth=3)**

**s = 'y = %s + %s \* x' %(str(gd\_w[0,0])[0:7], str(gd\_w[1, 0])[0:7])**

**ax.annotate(s, xy=(8.5, 9.3), xycoords='data',**

**xytext=(-180, 30), textcoords='offset points',**

**bbox=dict(boxstyle="round", fc='r', ec='r'),**

**arrowprops=dict(arrowstyle="->", fc='r', ec='r',**

**connectionstyle="angle,angleA=0,angleB=90,rad=10"))**

**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.legend(('training sampes', 'closed-form regression','gradient descent regression'),loc='upper center')**

**plt.show()**

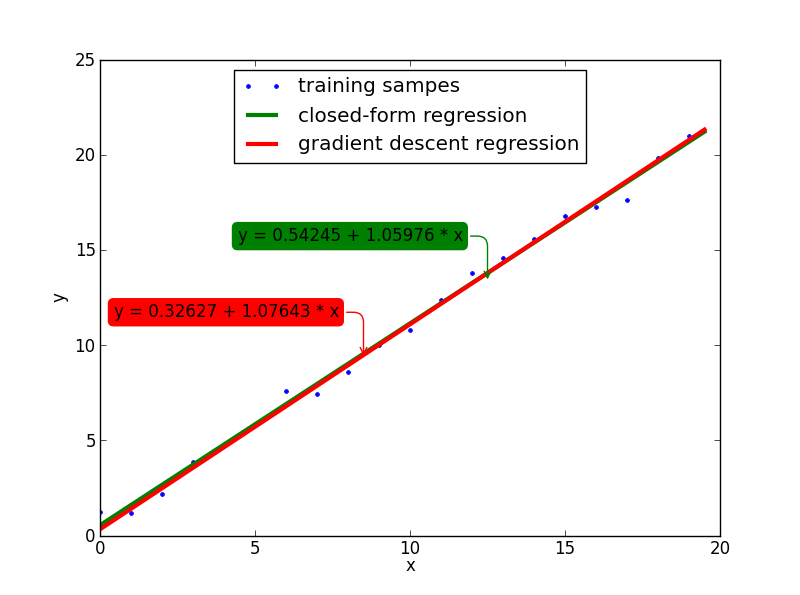


图2.1 解析解与梯度下降法解的对比图

## 3 基函数

### 3.1 多项式回归

如果有如图2.1的数据，依然采用式（1.2-1）的模型，则回归模型如图2.2。从图2.2中可以看出，用式（1.2-1）所表示的模型无法拟合这种带多个峰的数据。一个很直观的想法是增加式（1.2-1）中的项数，用多项式拟合这种多个峰的数据

（3.1-1）

式（3.1）写成矩阵形式为

（3.1-2）

按照1.2节中的方法，也可以得到的解，这里就不做详细推导，直接给出结论：

（3.1-3）

为了与式（1.2-7）加以区别，用代替了式（1.2-7）中的，这里的表示为如下形式

（3.1-4）

其中，为多项式中自变量的最高次数。

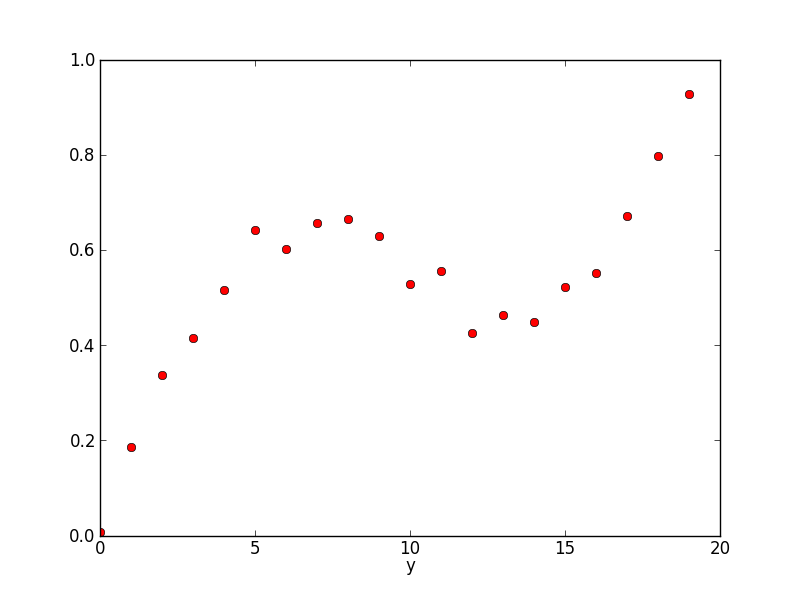


图3.1.1 加上随机噪声的数据

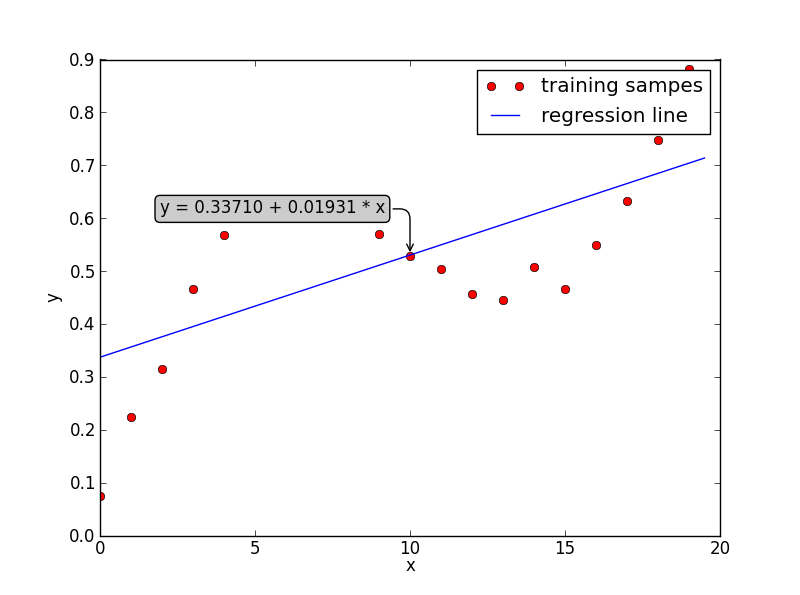


图3.1.2

按照式（3.1-3），分别用3、5、10、12阶多项式拟合数据，结果如图2.3。图（d）中的拟合曲线的末端上翘与数据不吻合了。这是过拟合导致的。过拟合问题将会在下一节中介绍。

实现图3.1.3的代码如下：

第一步：生成样本的代码

**def nlSamples(n = 100):**

**t = np.arange(0, 1.0, 1.0 / n)**

**y = [ti + 0.3 \* math.sin(2 \* math.pi \* ti)+random.random()\*0.01 for ti in t]**

**t = list(t)**

**return [t, y]**

第二步：按照式（3.1-3）计算模型的参数

**def bFLR(x, y, rank = 2):**

**x = np.matrix(x)**

**if x.shape[0] == 1:**

**x = x.transpose()**

**y = np.matrix(y)**

**if y.shape[0] == 1:**

**y = y.transpose()**

**one = np.ones((x.shape[0], 1))**

**tmp = np.zeros((x.shape[0], rank))**

**for i in xrange(rank):**

**tmp[:,i] = np.power(x.A, i + 1).transpose()**

**xx = np.hstack([one, tmp])**

**w = inv((xx.transpose()).dot(xx)).dot(np.transpose(xx)).dot(y)**

**return w**

第三步：用第三步中的参数画出拟合的

**def plotBFLR(w, x, y, rank = 2):**

**xx = [i for i in np.arange(0.0,1.0,1.0/20)]**

**w = w.A.transpose()**

**yy = [w.dot(xlist(i, rank))[0,0] for i in xx]**

**fig = plt.figure()**

**ax = fig.add\_subplot(111)**

**ax.plot(x,y, 'ro')**

**ax.plot(xx,yy)**

**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.title(str(rank) + ' order regression')**

**plt.legend(('training sampes','regression line'))**

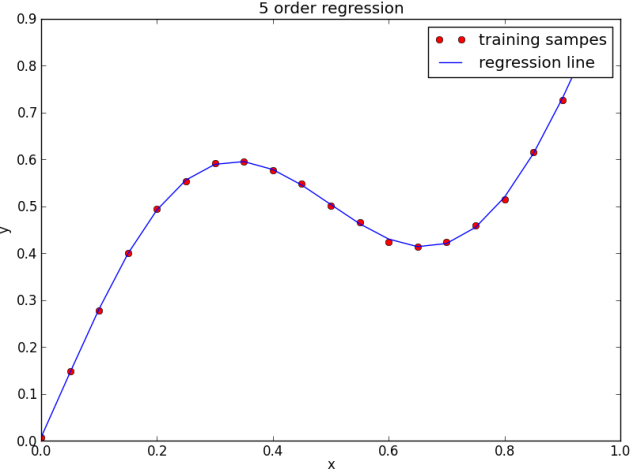
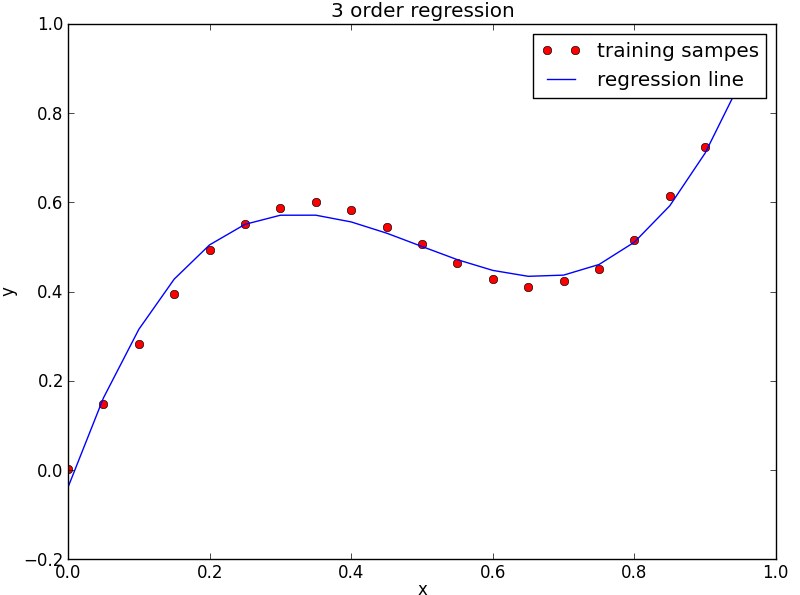
**plt.show()**

**def xlist(i, rank):**

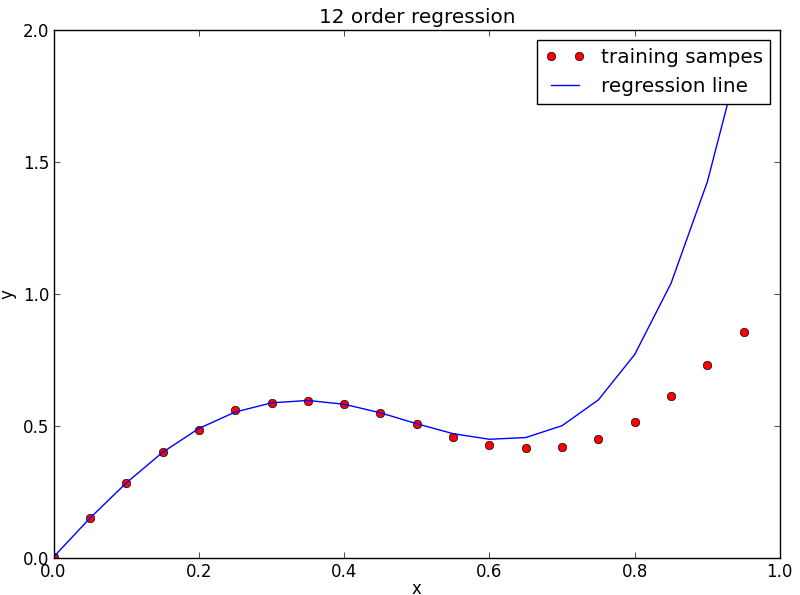
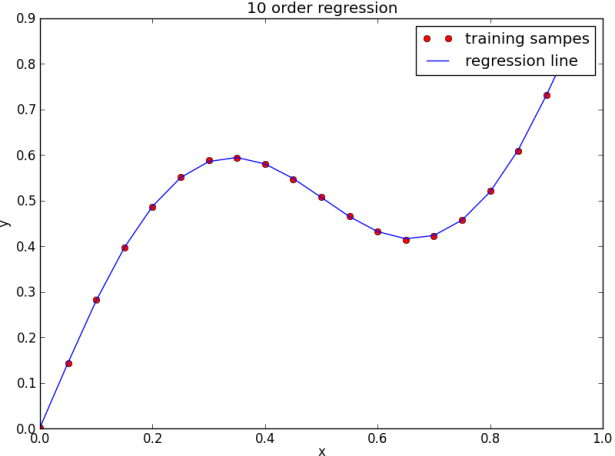
**l = [np.power(i ,ii) for ii in xrange(rank+1)]**

**l = np.array([l]).transpose()**

**return l**



（a）3阶多项式拟合结果 （b）5阶多项式拟合结果



（c）10阶多项式拟合结果 （d）12阶多项式拟合结果

图3.1.3 多项式拟合结果

### 3.2 回归模型中的基函数

式（3.1-1）更一般化的表示为

（3.2-1）

其中的称为基函数，引入基函数是为了对数据进行非线性变换，以解决非线性问题。

多项式回归中的都可看成是基函数。

其它形式的常用基函数有：高斯基函数，逻辑蒂斯基函数，它们的表达式分别如下

高斯基函数：

（3.2-2）

其中，控制着基函数的位置，控制着基函数的形状。

（3.2-2）

其中

## 4 欠拟合与过拟合

### 4.1 欠拟合

忽略严格的数学定义，从一般的直观理解欠拟合，概念如下。

**欠拟合**：模型过于简单，无法捕获数据中所存在的规律，图3.1.2所示的情况就是欠拟合，因为采用的模型为形式，这种形式的模型只能拟合x和y成线性关系的数据，对于非线性的数据，应该采用更高阶的回归模型。

**欠拟合的解决办法**：增加模型的复杂度，如将一次多项式模型，增加到3阶或者更高阶。对比图3.1.2和图3.1.3即可发现其变化过程。

### 4.2 过拟合

同样也可以给出过拟合的概念如下。

**过拟合：**和欠拟合相对**，**指模型过于复杂，模型在训练数据上的训练误差很小，而在测试数据的测试误差很大，即泛化能力很差。图3.1.3中的（d）就是过拟合现象，12阶的多项式模型对于（d）中的数据复杂度太高了，其实用3阶多项式模型就能取得不错的效果。

**过拟合的解决办法**：

1）不改变模型，增加数据

当过拟合时，不改变模型，增加数据可以改善过拟合问题，图3.1.3中的（d）只有20个数据点，现在将数据点增加到2000个，依然用12阶的多现实拟合，结果下图

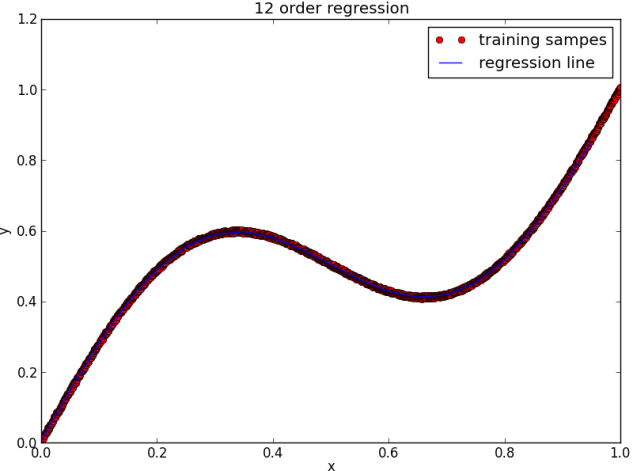


图4.2.1 增加数据点后的12阶回归模型

2）改变模型：**正则化**

依然对图3.1.3中的（d）的问题，如果没有足够的数据点，则可以减少模型中的特征，即将12阶模型降低为更低阶的模型，有一种方法称之为**正则化**，正则化方法通过在损失函数上加上罚项，对高阶项进行处罚，达到降低高阶项前面的系数，变小，说明模型中所对应的那项对模型的影响程度就会较低，达到简化模型的目的，常用的正则化后的损失函数为

（4.2-1）

按照前文介绍的方法，依然可以得到，正则化后的模型参数如下

（4.2-2）

其中，为罚参数，为单位阵。越大惩罚越强，如果非常大，则会趋向于0。

按照式（4.2-2），实现的代码如下

**def rTLR(x, y, lamda = 0.5,rank = 2):**

**x = np.matrix(x)**

**if x.shape[0] == 1:**

**x = x.transpose()**

**y = np.matrix(y)**

**if y.shape[0] == 1:**

**y = y.transpose()**

**one = np.ones((x.shape[0], 1))**

**tmp = np.zeros((x.shape[0], rank))**

**for i in xrange(rank):**

**tmp[:,i] = np.power(x.A, i + 1).transpose()**

**xx = np.hstack([one, tmp])**

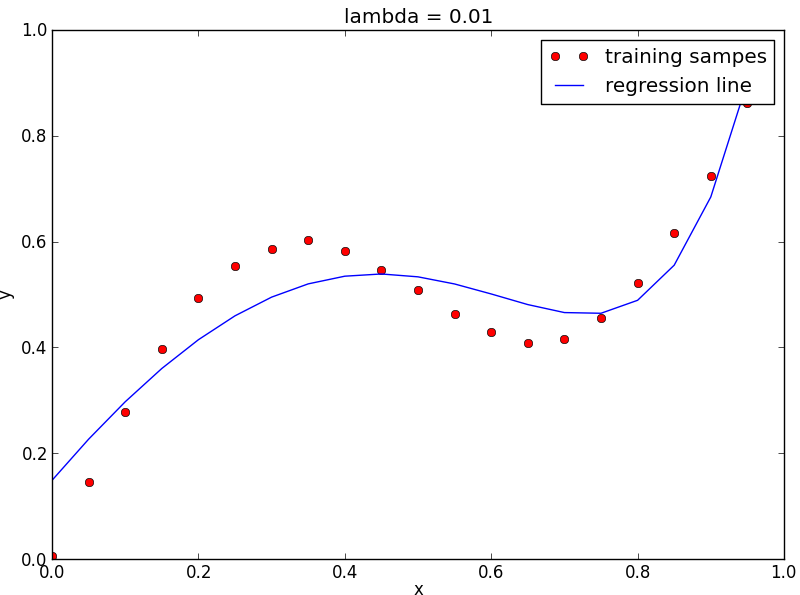
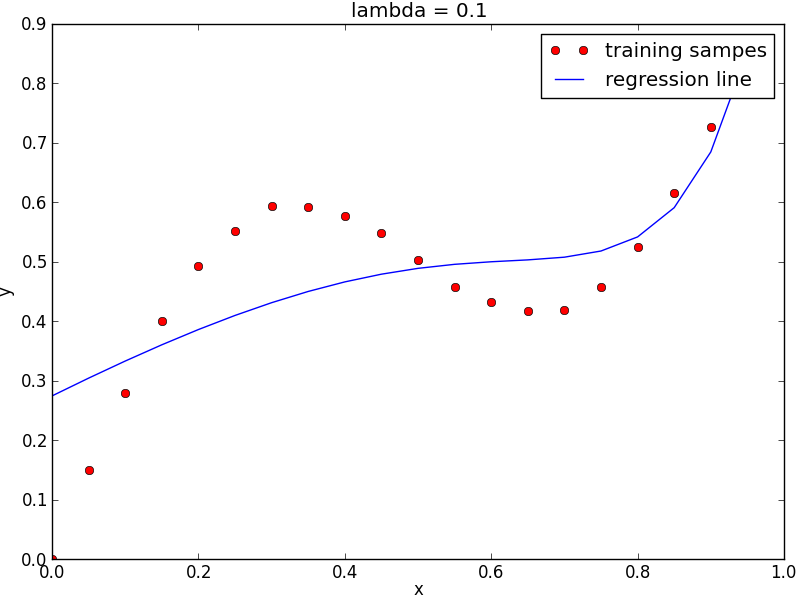
**dim = xx.shape[1]**

**I = lamda \* np.diag(np.ones(dim))**

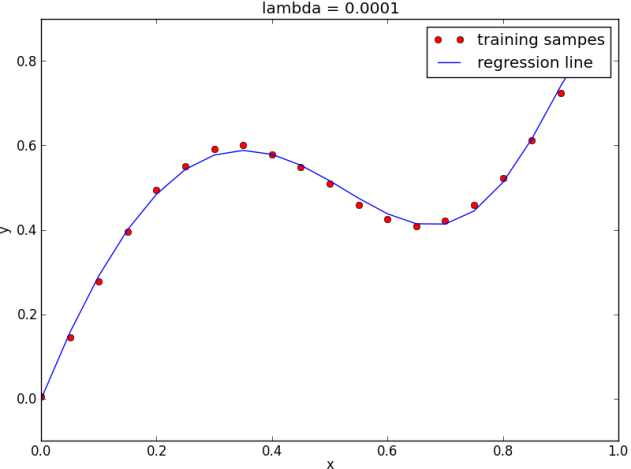
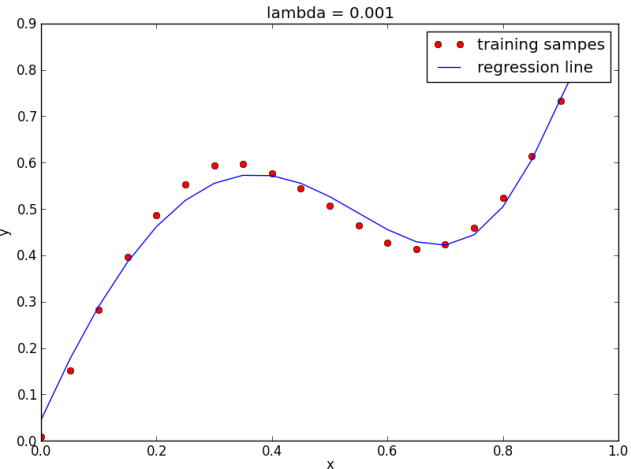
**w = inv(I + (xx.transpose()).dot(xx)).dot(np.transpose(xx)).dot(y)**

**return w**

图4.2.2是不同正则化参数下的回归曲线，从中可以看出越大，惩罚越强，时，多项式回归模型已经欠拟合了，对应于多项式中的高次项的系数接近于0了。随着变小，惩罚程度减弱，高次项的系数基本上没有变小，见（d）。



（a） （b）



（c） （d）

图4.2.2 不同罚参数下的拟合曲线

## 5 多元线性回归

以上介绍的是一元回归，如果自变量的个数不止一个，就会要求使用多元回归，多元线性回归最简单的形式如下

（5-1）

其中为自变量的个数。但是这种形式的多元回归模型有很多限制。

所以和一元回归类似，也可以引入基函数的概念，引入基函数后的表达如下

（5-2）

其中，，为基函数的个数

同理可以得到模型的参数为

（5-3）

其中

下面看一个二元回归的例子。

有如图5.1所示的曲面，建立一个回归模型拟合这个曲面，由于这个曲面是个二次曲面，所以在选择基函数时可以选择到二次或者更高次，比如选择，，此时

（5.4）

按照式（5.4）代入式（5.3）可以求得模型参数。

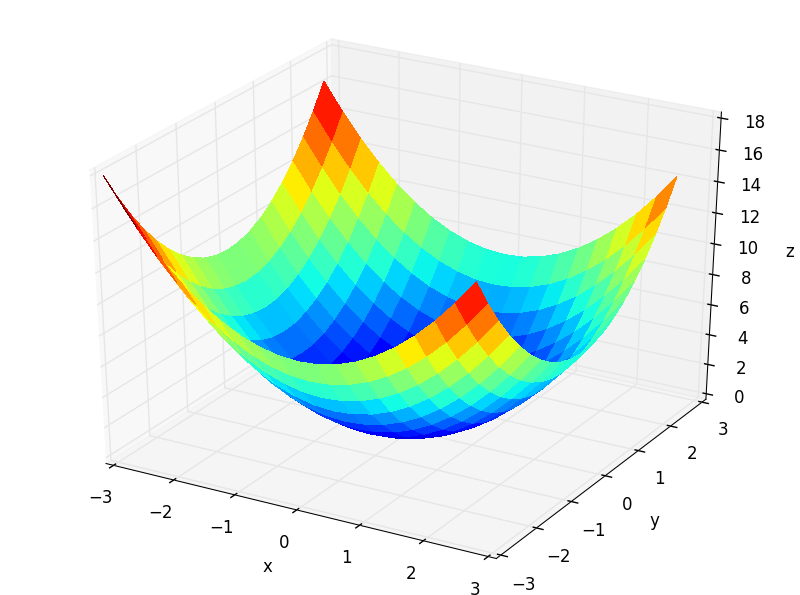


图5.1 二维曲面

下面看看用以上思路能否拟合出图5.1的曲面。

第一步：生成图5.1所示的数据，在*x*，*y*方向分别等距离采集数据点40个，Python代码如下：

**def xyz(n = 20):**

**t = np.arange(-3.0, 3.0, 6.0/n)**

**[x, y] = np.meshgrid(t,t)**

**z = x \* x + y \* y**

**x = np.reshape(x,(x.shape[0] \* x.shape[1], 1))**

**y = np.reshape(y,(y.shape[0] \* y.shape[1], 1))**

**z = np.reshape(z,(z.shape[0] \* z.shape[1], 1)**

**return [x.ravel(), y.ravel(), z.ravel()]**

第二步：用生成的数据点，按照式（5-3）求出参数

**def mrLR(x, y, z, rank = 5):**

**X = mrX(x, y, rank)**

**z = np.matrix(z)**

**if z.shape[0] == 1:**

**z = z.transpose()**

**w = inv((X.transpose()).dot(X)).dot(np.transpose(X)).dot(z)**

**return w**

**def mrX(x, y, rank):**

**a = xlist(x, rank)**

**b = xlist(y, rank)**

**one = np.ones((a.shape[0], 1))**

**c = np.hstack([one, a[:,:,0], b[:,:,0]])**

**return c**

第三步：用第二步中计算的参数，并且用第一步中的方法生成测试数据，在*x*，*y*方向上分别采样20个点，用这20个点作为测试数据，输入到模型中，看模型预测的值和真实的值之间的误差。图5.2中，将真实值和预测点按照对应的一行一行的展开，分别做成两个一维向量，这样便于比对。从图中看出，预测的结果还是精确的。

**def plotMR(w, x, y,z, rank = 5):**

**X = mrX(x, y, rank)**

**z = np.matrix(z)**

**if z.shape[0] == 1:**

**z = z.transpose()**

**z = z.A**

**z\_p = X.dot(w)**

**z\_p = z\_p.A**

**xx = np.reshape(x, (20, 20))**

**yy = np.reshape(y, (20, 20))**

**zz = np.reshape(z, (20, 20))**

**zz\_p = np.reshape(z\_p, (20, 20))**

**plt.plot(zz.ravel())**

**plt.plot(zz\_p.ravel(), '.')**

**plt.legend(('real data','prediction data'))**

**plt.ylabel('z value')**

**plt.xlabel('data point')**

**plt.show()**

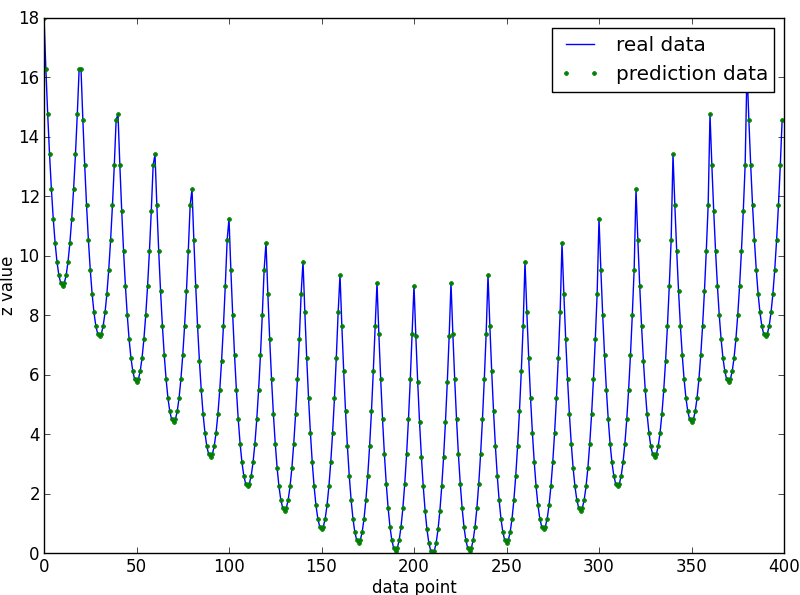


图5.2 真实值和预测值得对比图

实际上线性回归模型的非线性拟合能力较差，对于图5.3的多峰值函数就无能为力，本来这里给出的例子想用图5.3，结果，解释采用20阶以上的多项式也拟合不出图5.3的样子，无奈采用图5.1，相对简单一些，不过神经网络可以对高度非线性数据进行拟合，详细的Python代码和参考文献可以见

http://blog.csdn.net/zc02051126/article/details/9337319



图5.3 Matlab中的peaks函数产生的曲面

## 6 应用实例-Google票房预测模型

# 三 分类算法

## 1 线性分类器-感知器

### 1.1 感知器

有如图1.1所示的两类数据希望找到，如果想把他们分开，最简单的方法就是用图中的绿线将它们分开。显然绿线的方程为

（1.1-1）

假设红色点为，绿色点的集合为，则红色和绿色点分别满足

（1.1-2）

所以只要能求出式（1-1）并按照式（1-2）的规则就能将两类数据分开，将以上的想法表示成如下形式，可以称之为**感知器**

（1.1-3）

为符号函数

（1.1-4）

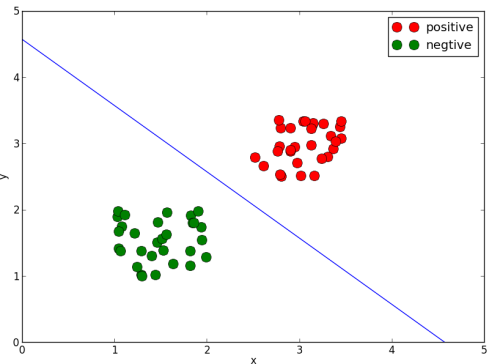


图1. 1

假设样本集合为，，为样本总数。

### 1.2 感知器的学习策略

和回归模型一样，感知器也要进行学习，才能获取感知器的参数，所以也需要构造**损失函数**。

图1.1中的直线称为分类线，如果数据是高维的，则线就会变成超平面了。在这里为了找到这条分类线，这条线需要满足这样的准则（**损失函数**）：**即线两边的点被误分的总数要最少，图1.1中所示的情况下，误分的总数为0。**这样的损失函数不是参数的连续可导函数，不容易优化，所以下面引入另一种损失函数。

对于误分类来说，有

（1.2-2）

这是因为，时，，反之亦然。因此将样本点中所有符合式（1.2-2）的样本点累加作为损失函数，并使其最小，就可以确定，损失函数如下

（1.2-3） （1.2-3）

其中的为个样本中被误分样本的个数。当式（1.2-3）为0时，全部样本分类正确。

### 1.3 优化损失函数

根据第二部分介绍的梯度下降法，需要求解损失函数的偏导数，式（1.2-3）可以直接求出偏导数而不用数值导数的求法

（1.3-1）

则

（1.3-2）

这里给出另一种类似梯度下降法的优化方法：**随机梯度下降法**（《Pattern Recognition And Machine Learning》M. Bishop一书中感知机一节中的解释），随机梯度下降法不对整个样本集合求导数，而对单个样本就导数，所以式（1.3-2）变成

最终的优化流程如下：

假设训练样本为，，为样本总数

（1）选取初值

（2）在训练集中选择

（3）如果，则更新：

（4）转到（2）直到训练集合中没有误分点

### 1.4 代码实现

第一步：生成训练数据集

**def twoSamples(n = 30):**

**x = []**

**y = []**

**x.extend([[1.0 + random.random(), 1.0 + random.random()] for i in xrange(n)])**

**x.extend([[2.5 + random.random(), 2.5 + random.random()] for i in xrange(n)])**

**y.extend([-1 for i in xrange(n)])**

**y.extend([1 for i in xrange(n)])**

**return [x, y]**

第二步：随机梯度下降法优化目标函数

**def perSGD(x, y):**

**w0 = np.array([0.1, 0.1])**

**b0 = 0.1**

**lamda = 0.1**

**while 1:**

**k = 0**

**for i in xrange(len(x)):**

**xi = x[i]**

**yi = y[i]**

**if yi \* (w0[0] \* xi[0] + w0[1] \* xi[1] + b0) <= 0.0:**

**w0 = w0 + lamda \* yi \* w0**

**b0 = b0 + lamda \* yi**

**k = k + 1**

**if k == 0:**

**break**

**return w0, b0**

第三步：画出分类线

**def plotDL(x, y, w, b):**

**px = [0.0, -b/w[1]]**

**py = [-b/w[0], 0.0]**

**x = np.array(x)**

**y = np.array(y)**

**plt.plot(x[y == 1, :][:,0], x[y == 1, :][:,1], 'ro', markersize = 12)**

**plt.plot(x[y == -1, :][:,0], x[y == -1, :][:,1], 'go', markersize = 12)**

**plt.plot(px, py)**

**plt.legend(('positive','negtive'))**

**plt.xlim([0, 5])**

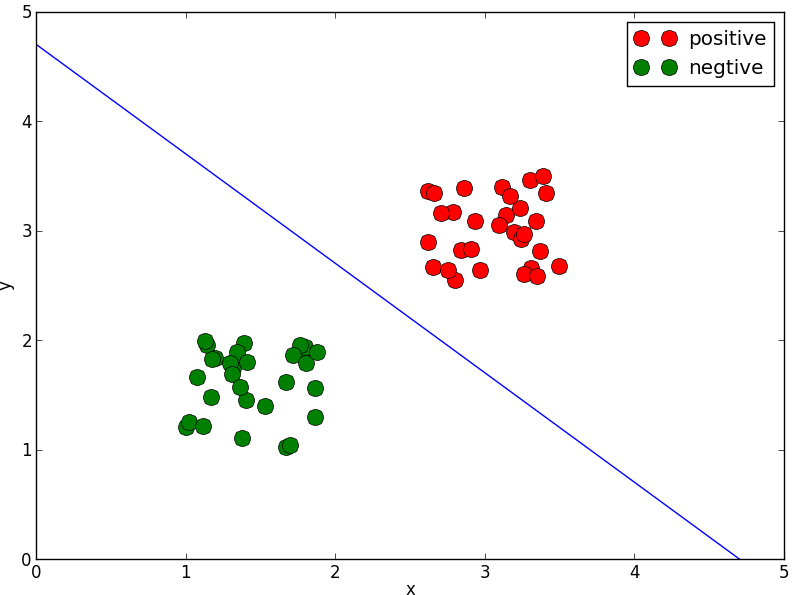
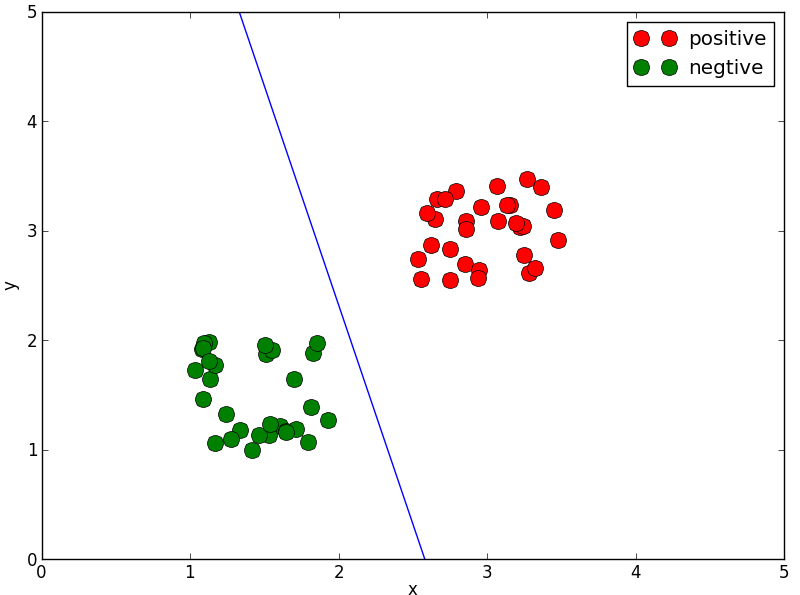
**plt.ylim([0, 5])**

**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.show()**

用上面的程序计算的两幅图如下



（a） （b）

图1.4.1 不同参数时的分类线

图1.4.1说明感知器对参数的初始值比较敏感，不同的初值得到不同的分类线。这也是感知器的缺点

## 2 线性分类器-逻辑回归

### 2.1 逻辑回归分布

定义：设为连续随机变量，服从逻辑分布是指具有如下分布函数和概率密度函数

（2.1-1）

（2.1-2）

其中，为位置参数，为形状参数。

下图是不用参数下的两幅分布图和概率密度图

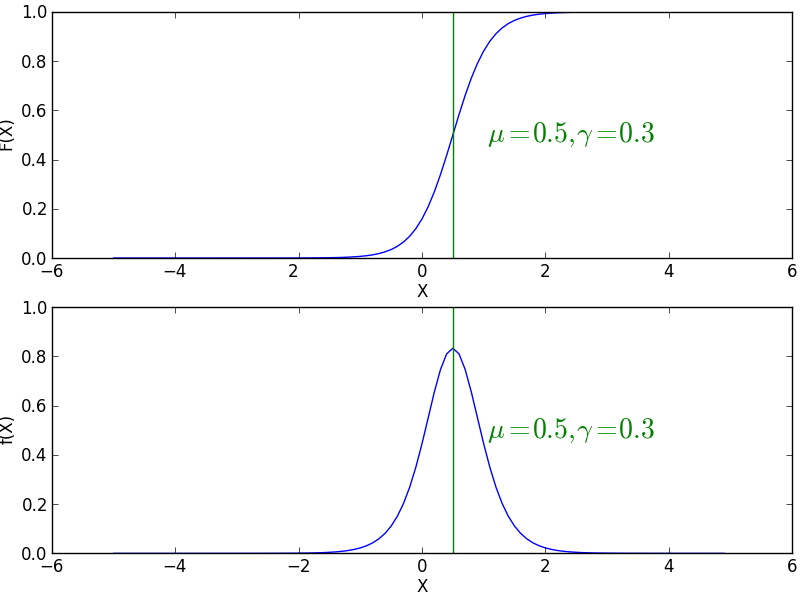
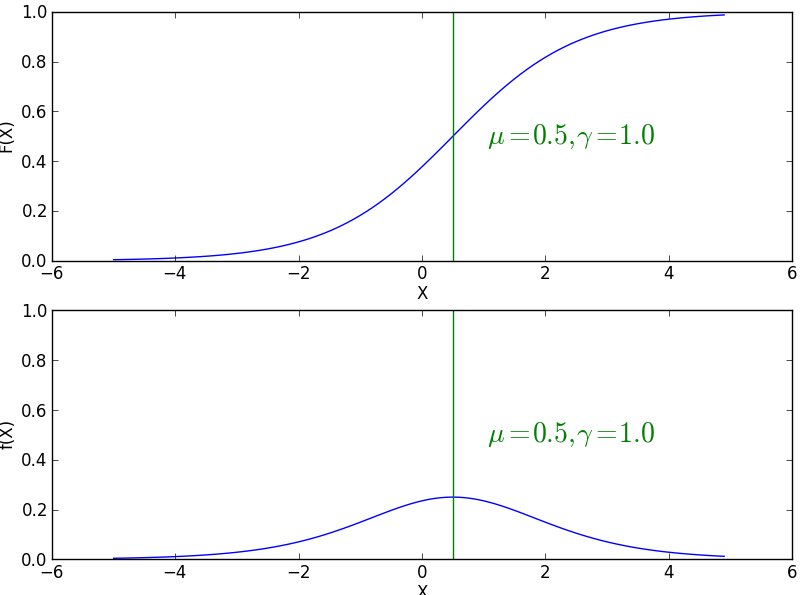


图 2.1.1 不同参数时的分布图和概率密度图

满足

（2.1-2）

对称中心为，随着变小在对称中心处变化率变大，变得尖锐。

### 2.2 二项逻辑回归

二项逻辑回归模型是一种分类模型，由条件概率表示，这里随机变量取值为实数，随机变量取0或者1，通过监督学习方法学习模型参数。二项逻辑回归的条件概率分布如下

（2.2-1）

（2.2-2）

这里，，称为权向量，称为偏置，称为与的内积。

分类时只要计算天健概率和哪个条件概率大，样本就属于哪一类。

### 2.3 参数估计

逻辑回归模型学习时，对于给定的训练集合，，，，可以用极大似然法估计模型的参数。按照极大似然估计的似然函数的定义，逻辑回归的似然函数为

（2.3-1）

因为是已知的量，所以似然函数可以省略掉，似然函数为

（2.3-2）

（2.3-3）

对数似然函数为

（2.3-4）

**按照极大似然估计理论，只要极大化对数自然函数，就可以估计出模型参数，**等**价于极小化负的对数似然函数，所以最优化方法最终的目标函数为**

（2.3-5）

式（2.3-5）的梯度为

（2.3-6）

只要按照下式迭代计算参数，当梯度满足一定的精度时，迭代结束，求得最终的最优参数

（2.3-7）

其中，为学习率（步长）。

### 2.4 基函数

和回归中的类似，分类中也能够引入基函数，在特征空间上对特征进行变换。只要将对数似然函数中的特征变量加上基函数，，式（2.3-5）和式（2.3-6）分别变为

（2.3-8）

式（2.3-5）的梯度为

（2.3-9）

### 2.5 过拟合（正则化）

为了解决过拟合问题，需要将式（2.3-8）加上一项罚项，变为

（2.3-10）

（2.3-11）

（2.3-12）

### 2.6 参数的矩阵表示

省略推导过程，给出带有基函数和正则项情况下的参数的矩阵表示

（2.3-13）

其中

是维的矩阵

### 2.7 代码实现

此节代码全部在softmax.py中，鉴于代码较多就不在文档中体现了。Softmax算法是逻辑回归的推广，即可以实现多分类的逻辑回归，详细的理论见<http://deeplearning.stanford.edu/wiki/index.php/Softmax%E5%9B%9E%E5%BD%92>。

Softmax.py中的类Softmax类有如下初始化参数：

算法的学习率： alfa（double）

算法的罚参数： lamda（double）

训练数据中的特征个数：feature\_num（int）

训练数据中的类别数目：label\_mum（int）

是否使用核函数： kernel （bool）

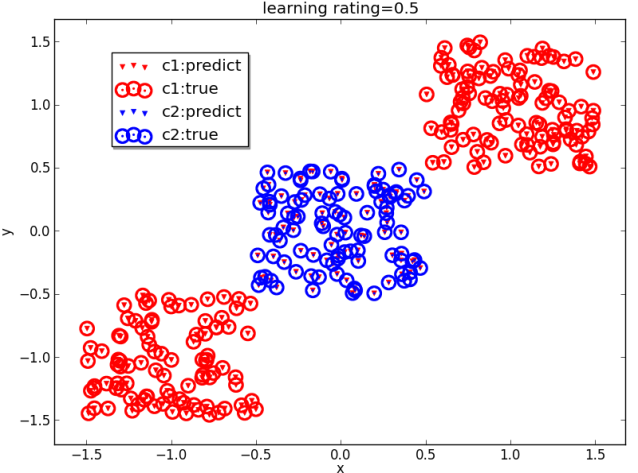
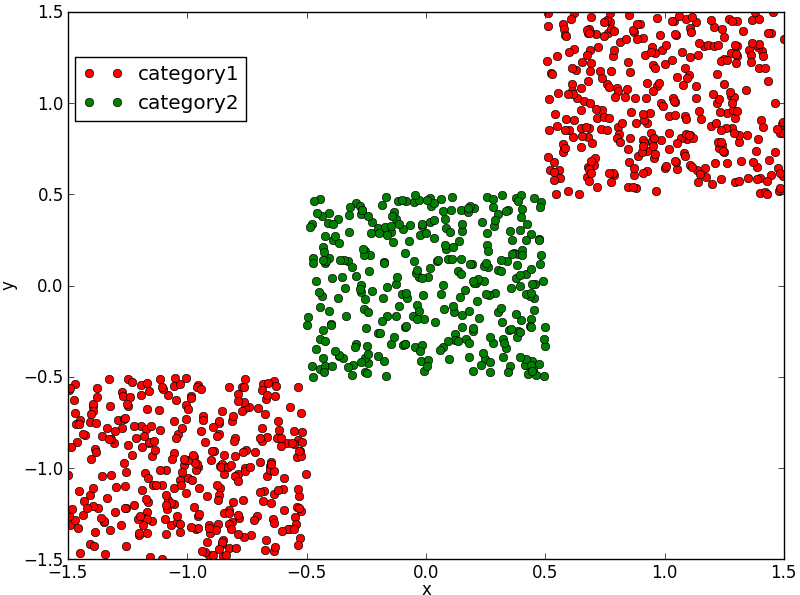
训练模型时的运行次数：run\_times = 500

训练模型时的收敛阈值： col = 1e-6

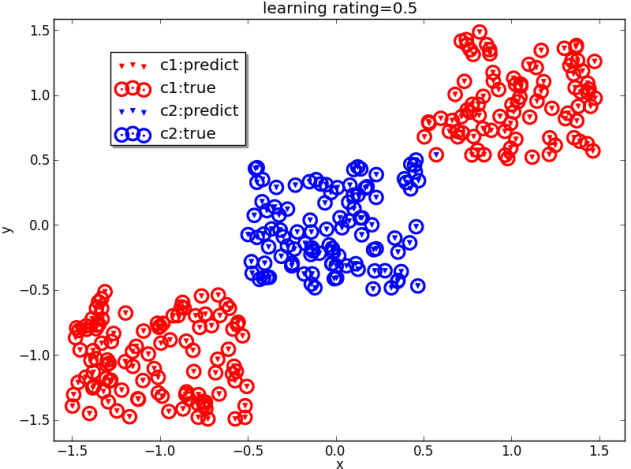
其中的train方法，接受两个参数，第一个为训练数据的特征，第二个参数为训练数据对应的标签

Predict方法接受一个参数为预测数据的特征，返回的结果为每个样本的分类结果。

图2.7.1中，（a）所示的数据是不可分的，（b）中显示了误分类的情况，（c）中展示了对图（a）中引入了基函数后的结果，显然图（c）中的数据是可分的了，分类结果见图（d）。



（a）原始的两类数据 （b）没用核函数的二分类结果



（c）加入基函数变换后的数据 （d）加入基函数后的分类结果

图2.7.1 二分类逻辑回归问题

图2.7.2是softmax对多分类的结果

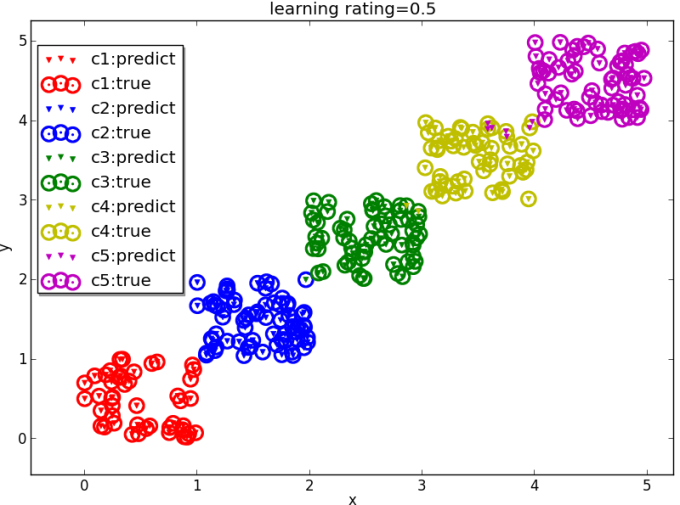


图2.7.2 softmax多分类结果

## 3 贝叶斯分类器

贝叶斯分类器，是最常用也是性能不错的分类器。其可以分为：基于高斯分布的贝叶斯分类器，主要对连续变量进行分类；多项式贝叶斯分类器，主要处理离散变量，如对文本的分类就是这种；贝努力贝叶斯分类器，主要处理二元样本。

### 3.1 贝叶斯公式

概率论里的贝叶斯公式如下

（3.1-1）

如果把当成类别，把当成样本，则式（5.1-1）可写为

（3.1-2）

表示样本属于类别的概率。**贝叶斯分类器的思想是，将样本属于每个类别的概率分别算出来，那个最大，样本就被分为哪个类别**。 假设此时有三个类别，则可以算出三个值分别为，，，由于最大，所以样本就属于**。**

重新研究式（5.1-2）发现对于同一个样本来说，比较时可以不考虑，因为对于同一个样本是一样的，所以式（5.1-2）又可以简化为

（3.1-3）

式（5.1-3）是实际应用中所使用的分类准则，下面将叙述如何计算连续样本和离散样本时的贝叶斯分类器。

### 3.2 高斯贝叶斯分类器

#### 3.2.1 理论简介

假设有训练样本集合，，为样本个数，，，按照3.2节中的介绍要想对一个未知的样本进行分类需要计算式（3.1.3）

即计算和

（3.1-4）

（3.1-5）

采用高斯分布形式，如下

（3.1-5）

为类别所对应的所有样本元素的均值和方差，，将展开写成如下形式

（3.1-6）

假设此时式（3.1-5）中的此时取，取为1，则此时将式（3.1-6）中X中的第一列中与所对应的项全部取出来，假设为

则此时

（3.1-6）

其它元素的计算方法与之类似。

#### 3.2.2 代码实现

第一步：生成训练样本和测试样本

**def samples(n\_samples, n\_features = 10, classes = 5, rat = 0.2):**

**见gaussian\_nb.py**

第二步：用训练数据训练模型

**def train(self, x, y):**

**见gaussian\_nb.py**

第三步：预测方法

**def predict(self, x):**

**见gaussian\_nb.py**

下图是朴素的高斯贝叶斯分类结果

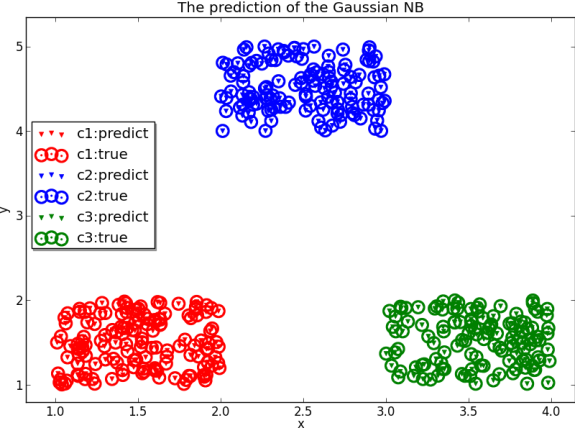


图5.2.2.1 高斯贝叶斯分类结果

### 3.3 多项式贝叶斯分类器

多项式贝叶斯分类器主要用来处理特征是离散的情况，例如其很善于对文本进行分类，线面将以实际的例子介绍多项式贝叶斯分类器在文本分类中的应用。

#### 3.3.1、构造数据集信息

假设存在如下五篇文档，并可以分成两大类good和bad

（1）'Nobody owns the water.','good'

（2）'the quick rabbit jumps fences','good'

（3）'the quick brown fox jumps','good'

（4）'buy pharmaceuticals now','bad'

（5）'make quick money at the online casino','bad'

`需要统计的信息如下

a、构造词语-类别矩阵

从每篇文章中解析出所有不重复的词语，把这些词语在每个类别中出现的次数记录下来形成表1的词语-类别矩阵

表1 词语-类别矩阵

|  |  |  |
| --- | --- | --- |
| 类别  词语 | good | bad |
| Nobody | Num11(1) | Num12(0) |
| owns | Num21(1) | Num22(0) |
| quick | Num31(2) | Num32(1) |
| ┆ | ┆ | ┆ |

表1表示在属于good类别的所有文章中，Nobody只出现在文章（1）中则Num11的值为1，在bad类别的所有文章中没有Nobody则Num12为0；而quick出现在文章（2）和（3）中所以Num31为2，同时quick出现在类别bad的文章（5）中一次，文章（4）中没有出现，所以Num32为1，其它的词语以此类推。

b、每种类别的个数

此处的文档共5篇，分为good和bad类，可以得到如表2的类别数统计信息

表2 类别信息

|  |  |
| --- | --- |
| 类别 | 数量 |
| good | N1(3) |
| bad | N2(2) |

#### 3.3.2、计算特征（单词）概率

有了1中的信息就可以计算概率了，表示在给定B的条件下A的概率。我们需要求的是，例如要计算，则计算方法如下



quick出现在good中两次，而属于good的文章为3篇，所以最终的结果为2/3。

更一般的



如果词语在表1中位于第i行，类别在表一中位于j列，则对应的类别在表2中位于j行，则



表示表1中*i*行*j*列对应的元素值，表示表2中对应的*j*行元素值。

这种计算概率的方法，在计算初期，信息量较小时可能会出现个小的问题，如单词*money*只出现在文档（5）中，并且这是一篇不好的文章，由于*money*只在一篇*bad*的文章中出现，而在任何good的文章中都没有出现，所以最后计算money在good文章中出现的概率为0。显然这样做有些偏激，因为money完全可能是一个中性词，只是其恰好出现在一篇bad的文章中而已。更合理的情况是随着单词越来越多地出现在一个分类文档中，对应的概率也会趋近于某一个数值。

为了解决这种缺点，引入加权平均方法，假设任意一个词语出现的概率为=0.5（也可以根据先验知识设置该值），其还需要一个权重，此处设为=1（也可以根据先验知识设置该值），同时为也取一个权值=*totals*（某一词语出现在所有分类中的次数（即，表1中某一行的元素相加，因为某一行对应于某一特征（词语）出现在*good*和*bad*类别中出现的次数）），最终加权平均的概率表达为



#### 3.3.3、计算整篇文档的频率

朴素的贝叶斯分类器假设每个特征（词语）的概率都是彼此独立的，所以计算某篇文档属于某一类别的概率，只需要将该篇文章中的所有不重复的特征属于某一类别的概率相乘，表达形式如下



其中N为某篇文章中所有特征（单词）的总数。

#### 3.3.4、贝叶斯公式

我们真正需要的是，贝叶斯公式是一种对条件概率进行调换求解的方法，其通常写作



在此处，可写为



在3中已经计算出来，为属于某一类别的文章总数除以全部文章的数量，在用贝叶斯进行计算分类时，需要把文章属于各个分类的概率都计算出来，然后比较，对于各个分类都是相同的所以不用计算，所以最终的表达式为



**注意：由于贝叶斯分类器需要将概率连乘，如果词项过多将可能导致计算机计算时溢出，所以可以将式**两端取对数，如下



因为对数是单调增函数，所以经过对数变换后就变成相加了，可以避免溢出。

#### 3.3.5、选择分类

在实际的应用中，为了保准分类的准确性，对于一篇将要被划分为某个分类的新文章而言，其概率与其它所有分类的概率相比，必须大于某个指定的数值，这个数值称为阈值，以垃圾邮件的分类为例，如果要把一份邮件过滤到bad分类中，设该邮件属于bad分类的概率为，属于good的概率为，只有满足时才能把这封邮件划分到bad分类中，否则就认为这封邮件是good分类。

#### 3.3.6、费舍尔方法

**1、针对词语的分类概率**

贝叶斯方法，将的计算结果组合起来（连乘）的得到了整篇文档的概率，然后再对其进行调换求解，费舍尔方法将直接计算当一篇文档中出现某个特征时，该文档属于某个分类的可能性，也就是计算，计算的方法如下



不过这种方法也会存在前文遇到的问题—算法接触到的词的次数太少，所以可能会对概率估计的不准，因此要像前文那样进行加权平均



**2、将各概率值组合起来**

费舍尔方法的计算过程是将所有的概率相乘起来，然后取自然对数，然后再将结果乘以-2



N为词语个数，*ln*是自然对数。

**3、对内容分类**

与贝叶斯分类器相似，为了使分类结果准确，要为每个分类指定个下限，而后分类器会返回指定范围内的最大值，例如，在垃圾过滤器中可以将bad的阈值设置很高为0.6，将good分类的阈值设置的较低为0.2，这样做就可以将正常邮件被分到bad分类中的可能降到最低，同时也允许少量垃圾邮件进入到收件箱中，如果有的邮件good分类的分值低于0.2，good分类的分值低于0.6，都被划分到未知分类中。

#### 3.3.7、增量式训练

在真实世界中所有的训练和分类都不可能一次性的完成，那么就需要将用户在训练期间所产生的与训练相关的数据保存起来，在下次训练的时候就不要重复训练了，这种支持分次训练的方式称之为**增量式训练，**在该算法中，每次训练时只要更新表1和表2的消息，并将之保存即可。

在分类时直接使用保存下来的表1和表2的消息就可以分类了。

## 4 总结

除了以上介绍的分类器外，还有神经网络、决策树、随机森林、Adaboost，支持向量机等，这里不做过多介绍。

分类算法属于有监督学习范畴，其需要先验的某些数据集对模型进行训练，用训练好的某些对未知数据进行分类，主要的流程如下：

1 将数据集（带标签的数据）按照一定的比例分成训练集和测试集，例如可以将80%的数据作为训练集，20%的数据作为测试集。

2 用训练集训练分类模型

3 用测试集测试模型的性能，如查看分类器的准确率召回率等评价指标，以确认模型的精度是否满足要求

4 用模型对未知类别的数据进行分类

## 5 应用实例-主动客服

# 四 聚类算法

聚类算法是无监督学习算法，对于没有打上标签的数据，可以采用聚类算法。下面介绍两种常用的算法KMeans和谱聚类。

## 1 Kmeans

假设有数据集合，，，如果想把该数据集合分成k个类，应该如何划分？这就是聚类问题。例如有如下二维数据（以二维为例方便展示）

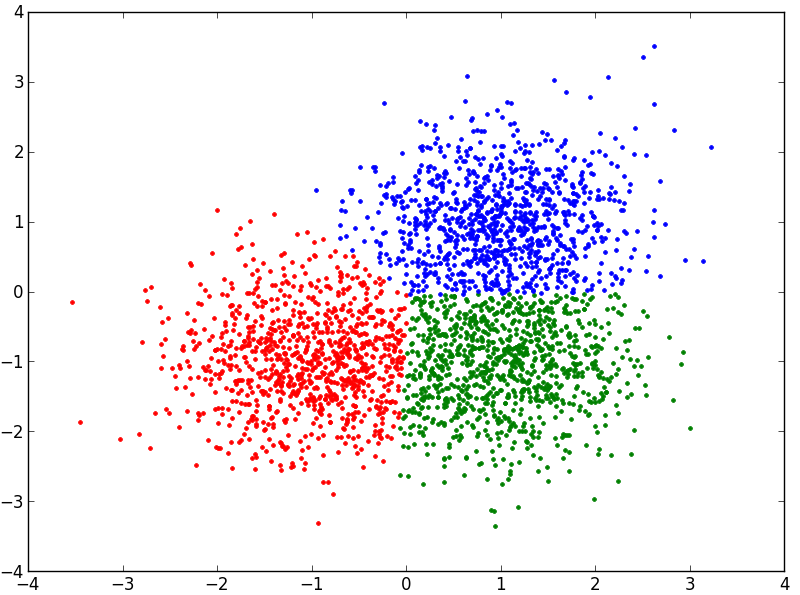


图1.1 三类数据

对于图1.1所示的数据可以使用KMeans方法对其进行聚类，KMeans算法是最简单高效稳健的算法，而且很容易**并行化**，但是其有个问题就是无法处理非凸集问题，处理非凸集问题的算法，将在下一节中介绍谱聚类将解决非凸集问题。

KMeans算法的流程如下：

（1）适当选择个类的初始中心，可以随机初始化；

（2）在第次迭代中，对任意一个样本，求其到个中心的距离，将该样本归到距离最短的中心所在的类；

（3）利用均值等方法更新该类的中心值；

（4）对于所有的个[聚类](http://baike.baidu.com/view/31801.htm" \t "_blank)中心，如果利用（2）（3）的[迭代法](http://baike.baidu.com/view/649495.htm" \t "_blank)更新后，值保持不变，则迭代结束，否则继续迭代。

对于以上的流程实现的代码如下

**def kmeans(data, k, tol = 1e-6):**

**data = np.array(data)**

**import random**

**index = np.arange(0, len(data))**

**random.shuffle(index)**

**index = index[0: k]**

**#(1)随机初始化初始的聚类中心**

**init\_center = data[index, :]**

**labels = np.zeros((data.shape[0],),dtype=np.int)**

**iter\_ = 0**

**while iter\_ < 100 :**

**err = 0.0**

**#（2）在k次迭代中将样本分配到最近的聚类中心中**

**for i in xrange(data.shape[0]):**

**di = data[i, :]**

**max\_dist = 1e100**

**tmp = []**

**for center in init\_center:**

**dis = pdist([di, center])**

**tmp.append(dis[0])**

**labels[i] = np.argmin(tmp)**

**#（3）计算新类的聚类中心**

**new\_center = []**

**for label in set(labels):**

**label\_data = data[labels == label, :]**

**new\_center.append(np.sum(label\_data, axis=0) / len(label\_data))**

**#（4）计算新的聚类中心和旧的聚类中心的差异**

**for i in range(0, k):**

**err = err + norm(new\_center[i] - init\_center[i])**

**init\_center = new\_center**

**iter\_ = iter\_ + 1**

**#画出每次迭代后的类别**

**observer(iter\_, data, labels, new\_center)**

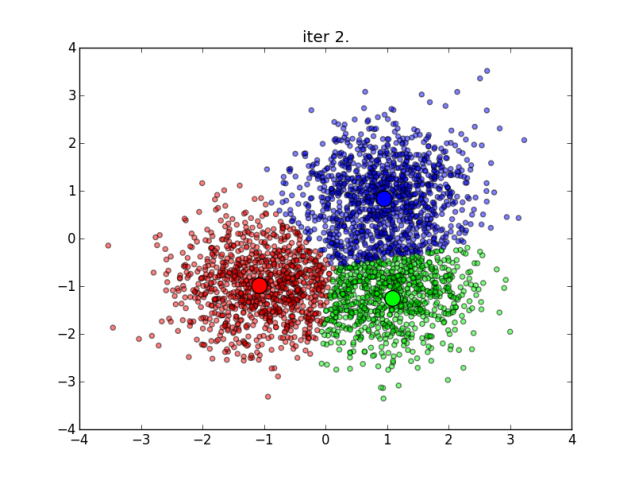
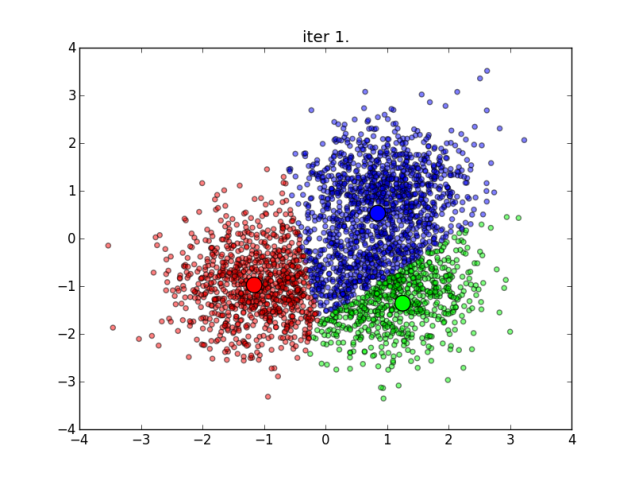
**#如果新的聚类中和旧的聚类中心的误差小于阈值停止迭代**

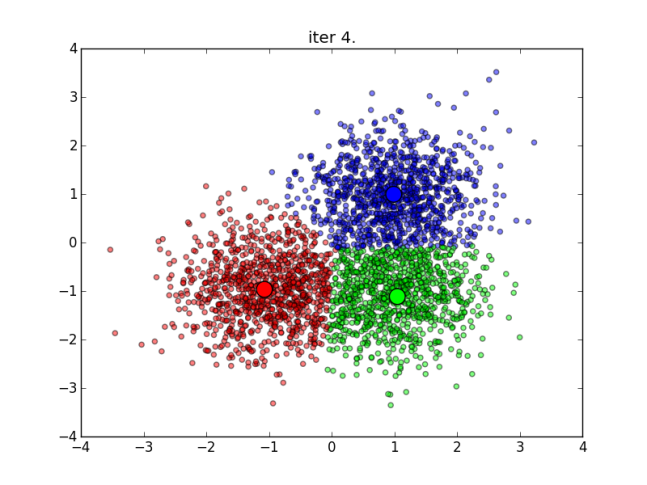
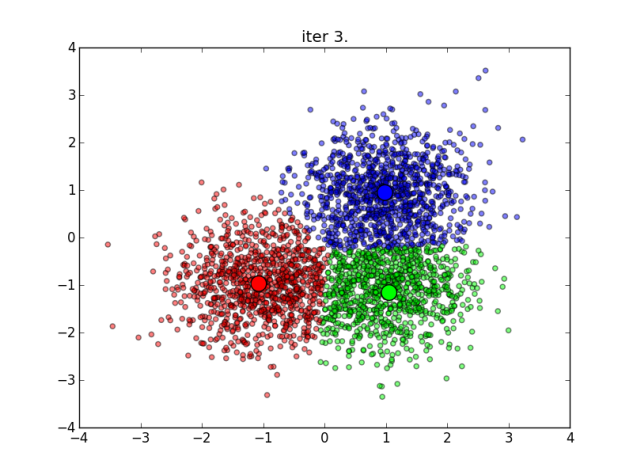
**if err < tol:**

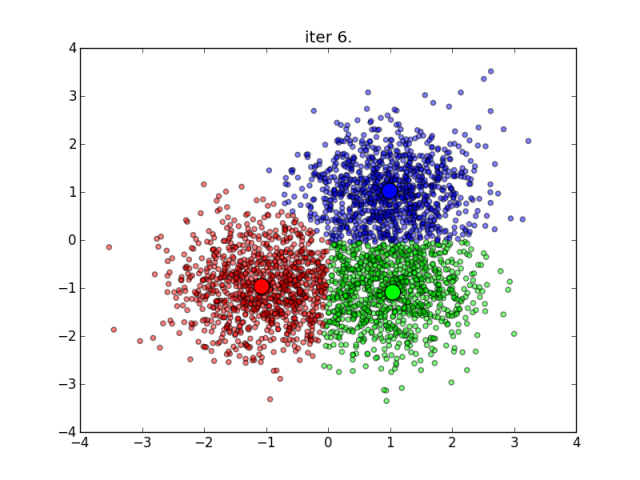
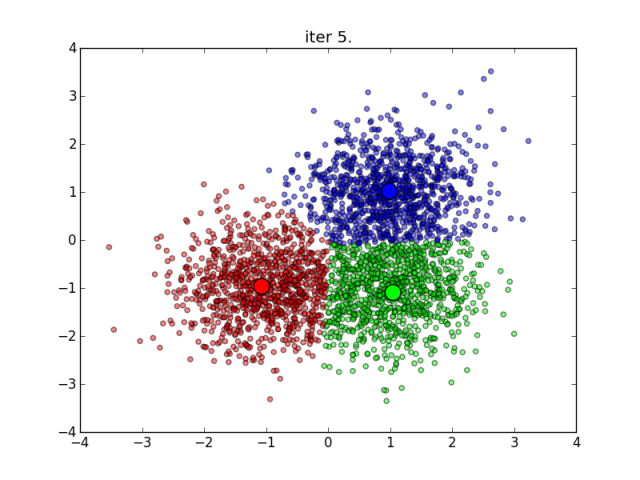
**break**

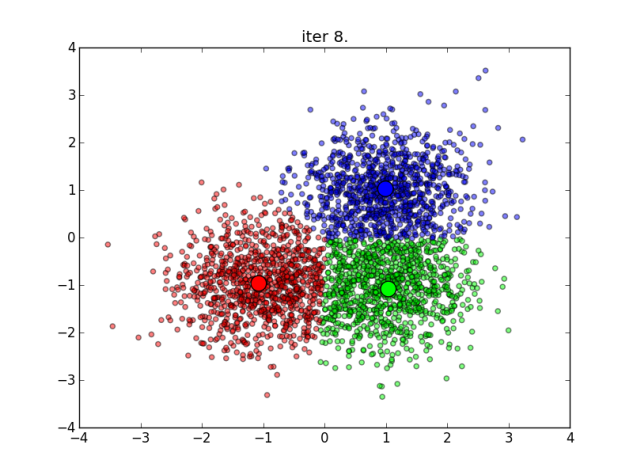
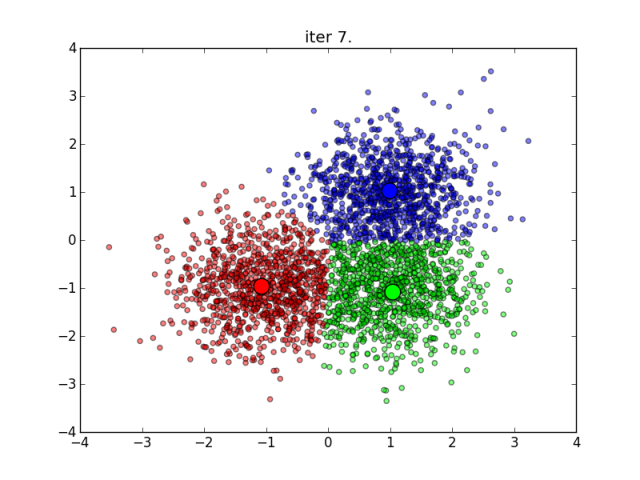
**return labels**

图1.2是图1.1经过十次迭代后的结果









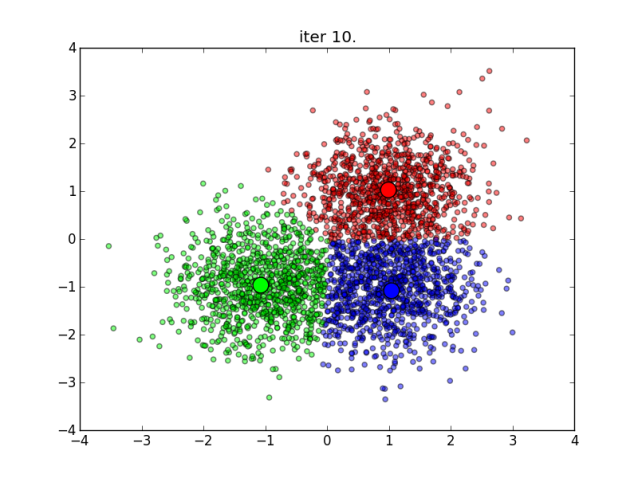


图1.2 KMeans十次迭代的过程图

## 2 谱聚类

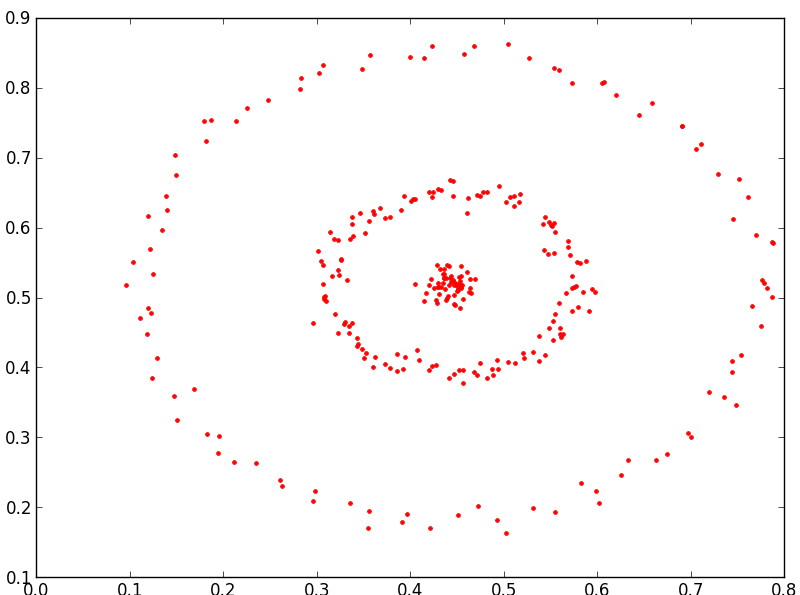


图2.1 谱聚类的原始数据

对于图2.1所示的数据KMeans就无能为力了，用KMeans对图2.1聚类的结果如下

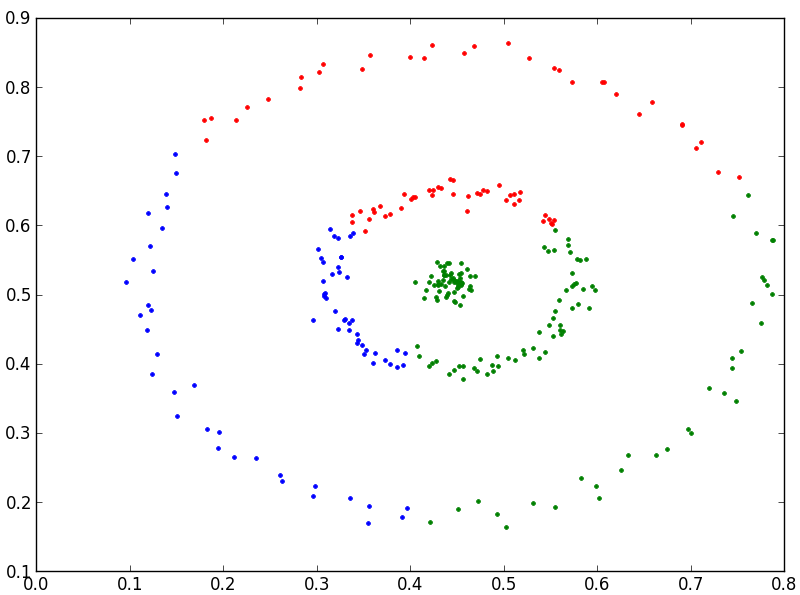


图2.2 KMeans对图2.1的聚类结果

下面看下谱聚类对图2.1的聚类结果

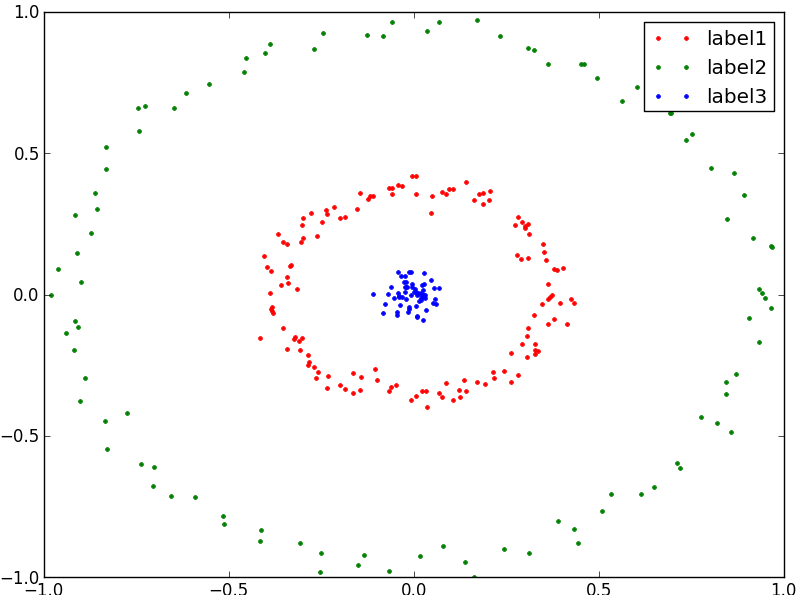


图2.2 谱聚类对图2.1的聚类结果

谱聚类的计算流程如下：

假设存在数据点集，将其聚成C类

（0）对S进行中性化

（1）构造相似矩阵，，，，为距离函数，通常为和的欧拉距离，为一重要的标量参数，下面会有详细介绍。

（2）构造对角矩阵*D*，，构造正规化的相似矩阵。

（3）选择一个期望聚类的数目*C*。

（4）找出*L*中最大的*C*个最大的特征值向量，构成矩阵

。

（5）将矩阵X的行向量单位化，得到矩阵，。

（6）将*Y*的每行看作属于的点，并通过*k-means*进行聚类。

（7）对*Y*进行聚类，聚类的结果就代表了原始数据点的聚类结果。

**谱聚类之所以能处理非凸集数据问题就是因为其对数据首先进行了变换，变换流程如流程中的（1）-（4），下面看下图2.1所示的数据经过（1）-（4）变换后的结果图**

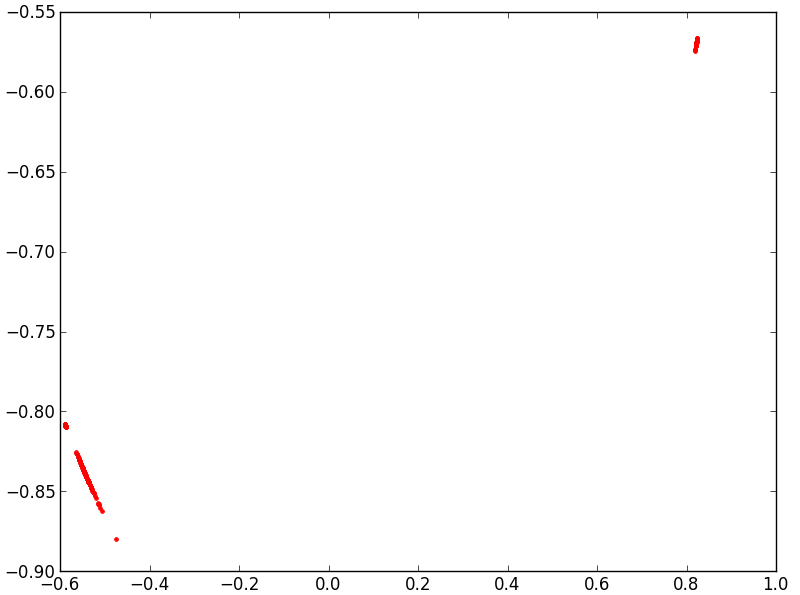
****

图2.3 经过（1）（4）变换后的数据

程序解析

第一步：加载数据

**def readData(path):**

**data = []**

**for line in open(path, 'r'):**

**ele = line.split('\t')**

**tmp = []**

**for e in ele:**

**tmp.append(float(e))**

**data.append(tmp)**

**return data**

第二步：对数据进行中性化，对应与流程中的（0）

**def normalize(data):**

**mean = np.mean(data, axis = 0)**

**data = (data - mean)**

**max\_ = np.max(np.abs(data))**

**data = data / max\_**

**return data**

第三步：计算相似矩阵，相当于（1）

：

**def affinity(data, knn):**

**data = np.array(data)**

**d = cdist(data,data,'sqeuclidean')**

**sigma\_list = np.zeros(data.shape[0])**

**dis\_matrix = np.zeros((data.shape[0], data.shape[0]))**

**i = 0**

**for e in d:**

**ec = copy.deepcopy(e)**

**ec.sort()**

**index = ec > 0.0**

**tmp = ec[index]**

**sigma\_list[i] = tmp[knn]**

**i = i + 1**

**dis\_matrix = updataAffinity(d, sigma\_list)**

**return dis\_matrix**

第四步：计算拉普拉斯矩阵，相当于（2）

**def lMatrix(dis\_matrix):**

**d = dMatrix(dis\_matrix)**

**if (len(d.shape) == 1):**

**d = np.array([d])**

**if(d.shape[0] == 1):**

**d\_row = copy.deepcopy(d.transpose())**

**d\_col = copy.deepcopy(d)**

**else:**

**d\_row = copy.deepcopy(d)**

**d\_col = copy.deepcopy(d.transpose())**

**tmp = d\_col \* dis\_matrix**

**l = tmp \* d\_row**

**return l**

第五步：计算拉普拉斯矩阵的奇异值，相当于（3）（4）

**def svd(l):**

**u,s,v = svds(l,2)**

**return u**

第六步：对奇异值分解的结果进行单位化，相当于（5）

**def unit(data):**

**norm2 = np.array([[norm(di) for di in data]]).transpose()**

**u = data / norm2**

**return u**

第六步：调用第一节中的KMeans算法对结果进行聚类，相当于（6）

**KMeans算法见上一节**

## 3 应用实例-网格化配送

# 五 降维算法

## 1 主成分分析（PCA）

### 1.1 主成分应用

从第二章的图1.1.1中能看出来哪部电影的人气指数最高吗？当然是票房比较高或者检索次数比较高的电影人气指数要高。如何用主成分来分析人气指数哪？沿着数据跨度最大的方向，即方差最大的方向画出一条坐标轴，然后将原始的数据投影在新的坐标轴上，投影后值越大代表综合人气指数最高，见下图。下面会以一个例子演示如何计算主成分，及其相关应用。

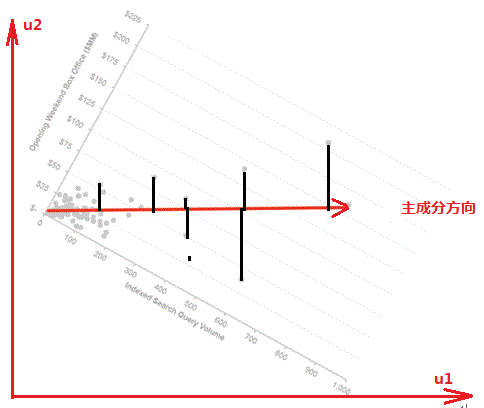


图1.1.1 各个样本点在主成分上的投影

### 1.2 一个例子

依然以电影为例，量表经常用于调查，例如在电影院看电影时遇到工作人员给你的一个量表，量表统计的内容包括：电影剧情，音乐，演员。经过打分后得到如下十部电影的评分结果

表1.1

|  |  |  |  |
| --- | --- | --- | --- |
|  | 电影剧情得分 | 音乐得分 | 演员得分 |
| 1 | 2 | 4 | 5 |
| 2 | 1 | 5 | 1 |
| 3 | 5 | 3 | 4 |
| 4 | 2 | 2 | 3 |
| 5 | 3 | 5 | 5 |
| 6 | 4 | 3 | 2 |
| 7 | 4 | 4 | 3 |
| 8 | 1 | 2 | 1 |
| 9 | 3 | 3 | 2 |
| 10 | 5 | 5 | 3 |

通过这个量表调查，如何能确定“**人气指数**”最高的电影？表1.1中的数据的三维图如下

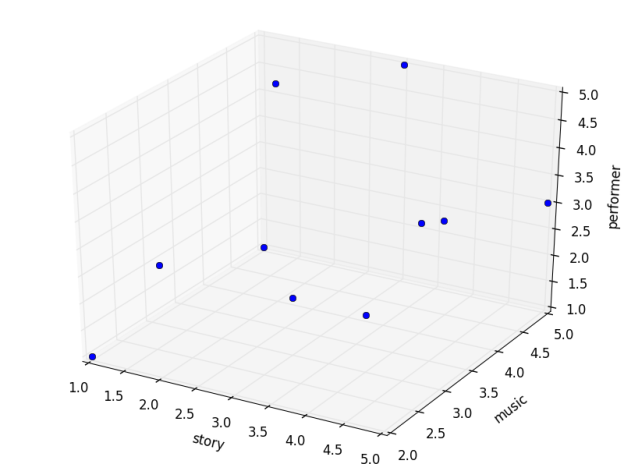


图1.2.1

下面忽略所有关于主成分的公式推导，直接给出求解步骤

#### 1.2.1 求主成分和主成分得分

**1 对数据进行标准化**

电影剧情得分，音乐得分，演员得分所对应的属性分别为，的均值和方差计算方法如下

（1.2.1-1）

的计算方法与之类似，经过计算后，表1.1的结果如下

表1.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 电影剧情得分 | 音乐得分 | 演员得分 |
| 2 | -0.67082039 | 0.3407771 | 1.44913767 |
| 3 | -1.34164079 | 1.19271985 | -1.31112456 |
| 4 | 1.34164079 | -0.51116565 | 0.75907212 |
| 5 | -0.67082039 | -1.3631084 | 0.06900656 |
| 6 | 0. | 1.19271985 | 1.44913767 |
| 7 | 0.67082039 | -0.51116565 | -0.621059 |
| 8 | 0.67082039 | 0.3407771 | 0.06900656 |
| 9 | -1.34164079 | -1.3631084 | -1.31112456 |
| 10 | 0. | -0.51116565 | -0.621059 |
| 11 | 1.34164079 | 1.19271985 | 0.06900656 |

**2 求相关矩阵**

表1.2的协方差矩阵如下

表1.3

|  |  |  |
| --- | --- | --- |
| 1. | 0.19050019 | 0.36004115 |
| 0.19050019 | 1. | 0.30048036 |
| 0.36004115 | 0.30048036 | 1. |

**3 求相关矩阵的特征值和对应的特征向量**

表1.4

|  |  |
| --- | --- |
| **特征值** | **特征向量** |
| **1.57285386** | **[ 0.571511 , 0.52211611, 0.63306393]** |
| **0.81400832** | **[ 0.60447096, -0.78960694, 0.105526 ]** |
| **0.61313782** | **[-0.5549685 , -0.32235949, 0.76687308]** |

**4 画出特征向量**

画出特征值**1.57285386和0.81400832所对应的特征向量**

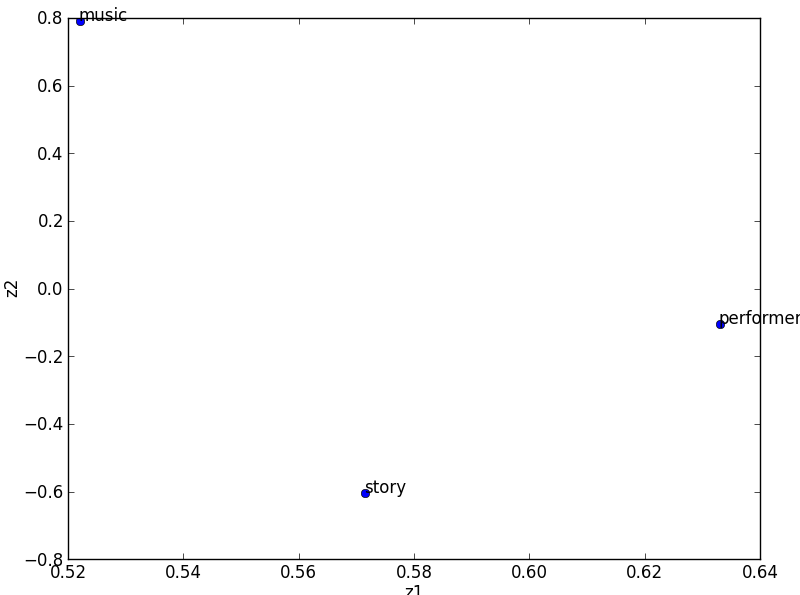


图1.2.1.1 特征在主成分坐标系下的散点图

**5 求出第一、二主成分**

**求第一二主成分的公式如下，**为原始数据的特征

所以求得的第一二主成分如下

表1.5

|  |  |  |
| --- | --- | --- |
|  | z1 | z2 |
| 1 | 0.711940768936 | 0.521649701201 |
| 2 | -0.974049885433 | 1.89112050144 |
| 3 | 0.98041582444 | -1.29470468257 |
| 4 | -1.05139653499 | -0.678110402438 |
| 5 | 1.54013504262 | 0.788858169925 |
| 6 | -0.276676639586 | -0.743573516808 |
| 7 | 0.604992012377 | -0.143693464988 |
| 8 | -2.30848899902 | -0.126979236679 |
| 9 | -0.660057875408 | -0.338082072825 |
| 10 | 1.43318628606 | 0.123515003736 |

第一二主成分画出的散点图如下

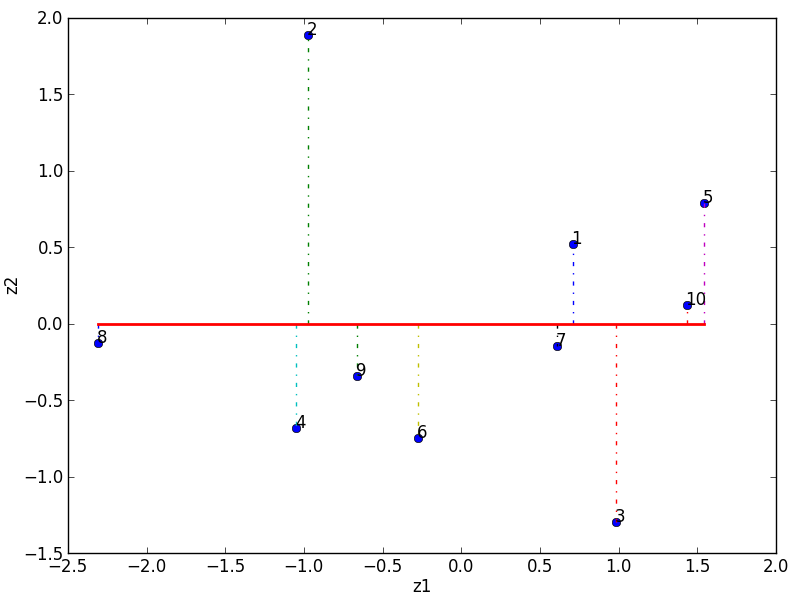


图1.2.1.2 第一二主成分的散点图

#### 1.2.2 确定分析精度

主成分分析的结果精度如何，是通过累积贡献率来确定的，第i个主成分的贡献率定义如下

本例中各个主成分的累积贡献率为

|  |  |
| --- | --- |
| **特征值** | **特征向量** |
| **1.57285386** | **1.57285386/3=52.42%** |
| **0.81400832** | **(0.81400832+1.57285386)/3=79.56%** |
| **0.61313782** | **(0.81400832+1.57285386+0.61313782)/3=100%** |

累积贡献率越大精度越高，一般只求第一二主成分，所以第一二主成分的累积贡献率越高越好，但是高到什么程度没有一个固定的标准。

#### 1.2.3 分析结果

第一主成分代表的物理意义为“综合性能”最好，在这个例子中就是“人气指数”

图1.2.1.2中表示的就是原始数据在第一主成分上的投影，投影越大“人气指数”越高，所以可以看出来5和10的人气指数最高，返回到表1.1中可以看出确实是5和10的人气指数比较高。

从图1.2.1.1中看出演员在对“人气指数”的影响最大。

**图1.2.1.2可以用于数据降维。**

#### 1.2.4 程序解析

第一步：对数据标准化处理，对应1.2.1节中第1步

**def unit(data):**

**mean\_ = np.mean(data, axis = 0)**

**std\_ = np.std(data, axis = 0, ddof = 1)**

**return (data - mean\_) / std\_**

第二步：计算协方差矩阵，对应于1.2.1节中第2步

**def cov(data):**

**mean\_ = np.mean(data, axis = 0)**

**data = data - mean\_**

**cov\_mat = data.T.dot(data) / (data.shape[0] - 1)**

**return cov\_mat**

第三步：直接调用scipy.linalg中特征值求解函数，求特征值，对应于1.2.1节中第3步

**[u, s] = eig(cov\_matrix)**

第四步：画出特征在新的坐标系下的散点图，对应于对应于1.2.1节中第4步

**def plotItems(u):**

**fig = plt.figure()**

**ax = fig.add\_subplot(111)**

**ax.plot(np.abs(u[:, 0]), u[:, 1], 'o')**

**u[:, 0] = np.abs(u[:, 0])**

**plt.text(u[0, 0], u[0, 1], 'story')**

**plt.text(u[1, 0], u[1, 1], 'music')**

**plt.text(u[2, 0], u[2, 1], 'performer')**

**plt.xlabel('z1')**

**plt.ylabel('z2')**

**plt.show()**

第五步：求出第一二主成分，并画出它们的散点图，对应于1.2.1节中第5步

**def samplesPca(data, u):**

**z1 = np.sum(data \* np.abs(u[:, 0]), axis = 1)**

**z2 = np.sum(data \* u[:, 1], axis = 1)**

**fig = plt.figure()**

**ax = fig.add\_subplot(111)**

**markerline, stemlines, baseline = ax.stem(z1, z2, '-.')**

**for i in range(0,z1.shape[0]):**

**plt.text(z1[i], z2[i], str(i+1))**

**plt.xlabel('z1')**

**plt.ylabel('z2')**

**plt.setp(markerline, 'markerfacecolor', 'b')**

**plt.setp(baseline, 'color','r', 'linewidth', 2)**

**u[:, 0] = np.abs(u[:, 0])**

**ax.plot(u[:, 0], u[:, 1], '\*')**

**plt.text(u[0, 0], u[0, 1], 'story')**

**plt.text(u[1, 0], u[1, 1], 'music')**

**plt.text(u[2, 0], u[2, 1], 'performer')**

**plt.show()**

## 2 隐性语意分析（LSA）

隐性语意分析是自然语言处理中常用的方法，其基于SVD，和PCA类似，又有所不同，在对文本的处理中，LSA直接对词-文档矩阵进行处理，而不是协方差矩阵，下面介绍LSA在文本处理中的两种应用：文本摘要，文本降维。

### 2.1 基于LSA的文本摘要算法

（1）假设存在文本D，将文本D分解为单个句子，句子的集合为S，为单个句子；并将文档D中所有的词提取出来，这些词的集合为T，,为词语。

（2）用S和T构建文档D的术语-句子词频矩阵A。下图中的每一行表示文档中的词语在每个句子中出现的次数，该矩阵的构造有多种方式，如直接统计词频，或者采用权重策略，详细的过程见文献（1）4.3节。

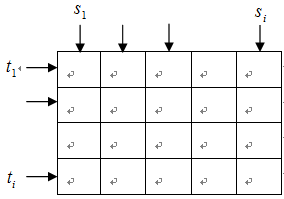


图1 术语-句子词频矩阵示意图

（3）对矩阵*A*进行奇异值分解（*SVD*），

如果



其中，是列正交矩阵，矩阵中的列向量称为左奇异向量；为对角阵它的对角线元素为从小到大排列的非负值；也为正交阵，它的列向量称之为右奇异向量，向量表示第*i*个特征中的句子，中每个元素表示句子在这个特征中的重要程度。

**如果**



**分解后*U*，*S*，*V*的物理意义如下（时）**



**注：**

如果，则满足：



可以从两个方面理解，将奇异值分解方法应用于术语-句子矩阵*A*的物理意义：（1）从变换观点看，*SVD*将*m*个术语-频率向量（*A*的行向量）张成的空间变换为*r*维线性无关的奇异矩阵向量空间。这种变换将*A*中的每列（代表句子*i*

所包含的术语频率）变换为矩阵的每列；同时把*A*中的行向量*j*（表示每个术语在每个句子中出现的频率）映射到矩阵的行向量。其中的每个元素，的每个元素分别称为*x*和

*y*个奇异向量的索引。（2）从语义的观点看，*SVD*可以获得矩阵*A*所代表的隐形语义结构，通过对A进行奇异值分解可以将A分解为*r*个线性无关的基向量或者称为概念，这些基向量或者说是概念可以表示文档中的术语和句子，SVD与传统的*IR*方法不同，*SVD*方法可以提取术语之间的相互关系，所以*SVD*方法可以对句子和术语的隐性语意进行聚类。比如有这几个词：doctor，physician，

hospital，medicine，nurse，这些词有相近的概念，同义词doctor和physician出现在相似的上下文中时，可能会有很多相关的词如：hospital，medicine，nurse和它们相联系。由于这些相似的词之间的结合模式，经过计算，就可以把doctor和physician映射到r维的奇异向量空间中相近的位置；除此之外，在一篇文档中，如果词的组合模式很重要且经常出现，那么这种特性也可以通过奇异向量表示，在奇异向量中的每个元素的幅值就表示了这种特性在文档中的重要性，任何句子，如果包含这种词语结合模式的特性，其将被映射到同一个奇异向量中去，奇异向量中的最大值代表的就是这个句子。因为在文档中每个特定词的组合模式都表示一定的主题或者概念，所以以上所讨论的事实将会导出如下情形：每个奇异向量代表一个主题或者概念，而奇异向量所对应的奇异值的大小表示可隐性语意的重要性。

（4）从中取出第i个列向量。

（5）从的第i个列向量中选出最大的值所对应的句子作为摘要。

（6）如果选择的句子数目已经达到了要求用户的要求，则停止计算。

### 2.2 文本降维

在2.1中看到经过分解的U矩阵如下，在文本聚类中原始的矩阵“词语-文档”可能会非常稀疏，在计算余弦相似性时导致计算的精度下降，所以可以采用SVD对其分解，然后对U矩阵进行聚类，因为U矩阵中的每行还是代表词语，只不过每列代表了隐性的语意，而且U矩阵是稠密的在计算余弦相似性是会比稀疏的效果更好。



## 3 应用实例-文本聚类

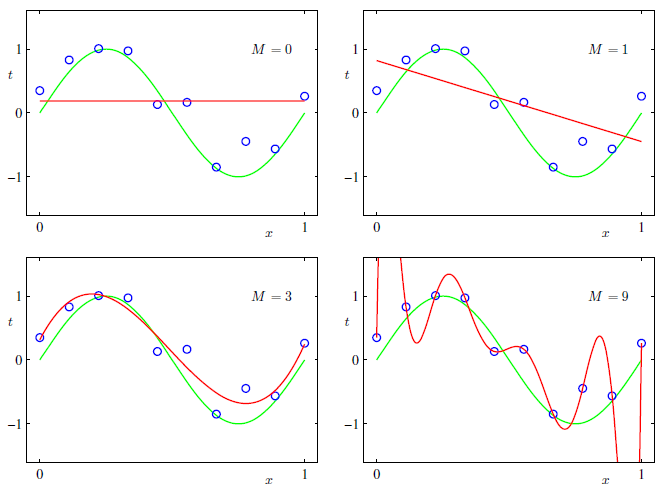
下面采用sklearn中的一个例子，详见[http://scikit-learn.org/stable/auto\_examples/document\_clustering.html#example-document-clustering-py](http://scikit-learn.org/stable/auto_examples/document_clustering.html" \l "example-document-clustering-py)

对文本进行聚类，可以看到在采用降维后聚类的准确率从55%左右提高到了75%左右

# 六 模型选择

## 1 模型选择方法

下图是不同阶多项式回归问题，从中可以看到不同的M取值所对应的不同效果，M=0和M=1时都是欠拟合，M=9多拟合，M=3时多项式回归模型刚好合适，在实际应用中不能将每个模型都画出来进行观察。



在模型选择中如何才能得到一个不过拟合也不欠拟合的模型？将数据集分为：训练集，交叉验证集，测试集。对于每个可能的模型，用训练数据训练模型然后在训练好的模型上做交叉验证，在所有的模型中选出交叉验证结果最好的模型最为最终模型，然后在最终模型上的测试数据的精度作为最终的模型精度。

## 2 交叉验证

**有的时候也将数据集分为训练集和测试集**，将训练数据集再次作为测试数据集用于测试模型的性能，在方法论上是错误的，因为这种测试方法会使模型对未知数据的预测误差变得很大。

为了避免过拟合，用训练集训练模型，用测试集测试模型的性能。

但是这种划分方法会减少训练集中的样本，也许对模型很有用的样本最终就没有被划分到训练集中。所以一种解决方法是将数据全集连续划分多次，使之成为多个训练集和测试集，最终的模型得分是这几个训练集和测试集的平均得分，这一过程称为交叉验证。交叉验证计算时间较长，但是不会浪费样本数据信息，交叉验证方法在小数据时相当有用。

**2.1 计算交叉验证指标**

有数据*d*，经过*k*次平均分割，每次分割成和分别代表第个测试集合训练集，用训练模型，并用测试模型，得到的第个模型的得分是，当结果次执行后，共得到个得分，求这个得分的平均值作为模型的最终得分。

### 2.2 数据集分割方法

**2.2.1 K折法**

将数据平均分成K份，称为K折。用其中的K-1份作为训练数据，剩下的一份作为测试，进行K次训练和测试，得到K个得分作为最终的得分。

**2.2.2 留一验证法（LOO）**

留一验证从样本全集中取出一个作为测试样本，剩下的所有样本作为训练样本。假设有个样本，将会有个不同的训练集和测试集，这种方法不会浪费过多的训练数据，因为只有一个数据作为测试样本。

**2.2.3 留P个样本验证（LPO）**

这个方法和留一验证法类似，留一验证中留的是留一个样本作为测试样本，这里留的是个样本而已。

## 3 模型性能的评价准则

### 3.1 混淆矩阵

二分类问题的混淆矩阵如下，为预测结果。



图3.1.1 二分类问题混淆矩阵

定义：

TP：为正类别预测为正类（正确预测）；FP：正类被预测为负类（错误预测）；FN：负类被预测为正类（错误预测）；TN：负类被预测为负类（正确预测）。

为预测结果

### 3.2 准确率、召回率、F-得分

**1 二分类问题**

定义：

按照图1.2-1中的定义，得到二分类的准确率、召回率、F-measures如下

**2 多分类问题**

定义：

* 是预测结果（集合），集合元素为
* 是真实结果（集合），集合元素为
* 是类别集合
* 是样本集合
* 是类别的预测结果

1）对所有类别的准确率、召回率、度量值求平均：

2）对整个样本计算准确率、召回率、度量值：

这中算法中

3）支持度加权平均计算准确率、召回率、度量值：

4）分别计算每个类别的准确率、召回率、F-得分，类似于1）中的不加权情况。

# 七 推荐算法

## 7.1、推荐系统概述

如何在学习爆炸的时代找到自己需要的信息？搜索引擎（Google，Bing，百度等等）成为大家快速找到目标信息的最好途径。在用户对自己需求相对明确的时候，用搜索引擎很方便的通过关键字搜索很快的找到自己需要的信息。**但搜索引擎并不能完全满足用户对信息发现的需求，那是因为在很多情况下，用户其实并不明确自己的需要，或者他们的需求很难用简单的关键字来表述。又或者他们需要更加符合他们个人口味和喜好的结果，因此出现了推荐系统，与搜索引擎对应，大家也习惯称它为推荐引擎**。

随着推荐引擎的出现，用户获取信息的方式从简单的目标明确的数据的搜索转换到更高级更符合人们使用习惯的信息发现。

推荐引擎的原理图如下



图7.1.1 推荐引擎工作原理

## 7.2、推荐系统类型

推荐系统主要可以分为：**协同过滤型和基于内容的推荐系统**，本文主要讨论协同过滤型推荐系统的各个相关模块，并且是基于开源框架Mahout实现的。基于内容的推荐不在本文讨论范围。

## 7.3、推荐系统的组成

推荐系统可以分为三个模块：算法测试模块；推荐线上运行时模块；推荐系统线下训练模块，下面分别介绍。



## 7.4、基于协同过滤的推荐

### 7.4.1 什么是协同过滤

**协同过滤是利用集体智慧的一个典型方法**。要理解什么是协同过滤 (Collaborative Filtering, 简称 CF)，首先想一个简单的问题，如果你现在想查找专业知识，但你不知道如何查找，你会怎么做？大部分的人会问问周围的朋友，看看他们是用什么方法解决你想解决的专业问题的，而我们一般更倾向于从口味比较类似的朋友那里得到推荐。这就是协同过滤的核心思想。

协同过滤一般是在海量的用户中发掘出一小部分和你品位比较类似的，在协同过滤中，这些用户成为邻居，然后根据他们喜欢的其他东西组织成一个排序的目录作为推荐给你。当然其中有一个核心的问题：

如何确定一个用户是不是和你有相似的品位？

如何将邻居们的喜好组织成一个排序的目录？

协同过滤相对于集体智慧而言，它从一定程度上保留了个体的特征，就是你的品位偏好，所以它更多可以作为个性化推荐的算法思想。

### 7.4.2 协同过滤的核心

要实现协同过滤，需要以下几个步骤

* 收集用户偏好
* 找到相似的用户或物品
* 计算推荐

协同过滤分为两种：基于用户的协同过滤（user-based）；基于项目的协同过滤（item-based）

### 7.4.3 协同过滤算法

#### 4.3.1 基于用户的协同过滤推荐

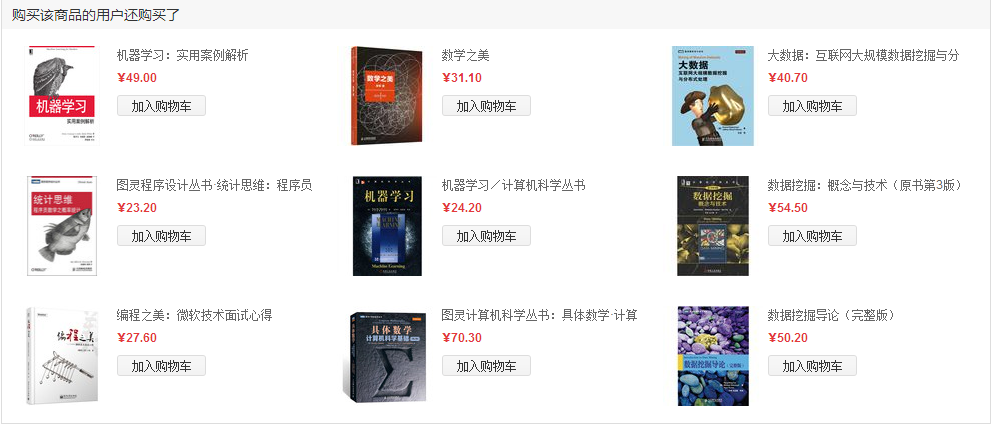


图4.3.1 User-based推荐

假设每个用户的购买记录格式为

表4.3.1.1 用户的购买记录

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 商品id  用户id | i1 | i2 | i3 | i4 | i5 | i6 | | i7 | i8 |
| u1 | 1 | 1 | 1 | 1 | 1 | 0 | | 0 | 0 |
| u2 | 1 | 1 | 1 | 1 | 0 | 1 | | 1 | 1 |
| u3 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | | 1 |
| u4 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | | 0 |
| u5 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | | 0 |

以上表中的数据，说明User-based推荐算法的实现过程，表中每一行数据代表了一个用户曾经购买过的商品记录，1代表购买过，0代表没购买过。

1）为每个用户寻找相似的其它用户，表示第i个用户与第j个用户的相似性，相似性的计算方法有多重，如欧式距离，余弦相似度，皮尔逊相关度等，在这里不做详细介绍，在这个例子中，以寻找u1的相似用户作为例子，其它的用户与之类似。为了便于理解，表中灰色部分显然可以计算相似性，因为对于u1来说i6-i8为0所以它们对计算相似性无贡献。这里采用购买过同一个商品的数量作为相似性。显然u1与u2-u6的相似性分别为：4,3,2,1。

2）查找只有u1的相似用户中购买过的商品，而u1没有购买过的商品，例如u2中的i6-i8，u1就没购买过；同理u3的i6-i8，u1也没购买过，u4中i6,i7是u1没购买过的，u5中的i6是u1没购买过的。

3）引入这样的假设**“人以类聚，物以群分”。**既然是相似用户，所以就有相似的兴趣或者购买习惯。那么就可以将u2-u5购买过而u1没购买过的商品推荐给u1。在2）中已经找出了u1没购买过的商品为i6-i8。对于这三个商品可以推荐给u1了。注意：购物网站的商品可是有千万种，如果商品过多的话，是不是全部要推荐给用户哪？当然不是，而是要选取最“靠谱”的商品推荐给用户，这里的“靠谱”就是要对商品进行排序了。

4）对推荐结果进行排序，i6-i8的得分计算方式分别为：，，，这很容易理解，因为对于i6来说，u2-u5都购买了所以应该排在前面，i7和i8亦然。

#### 4.3.2 基于项目的协同过滤推荐算法



图4.3.2.1 Item-based推荐

在上一节中介绍了User-based算法，其是通过寻找相似的用户达到推荐的目的。如果想找到与某一商品相类似的其它商品又该如何实现？Item-based算法可以实现这一目的，将表4.3.2.1转置变为表4.3.2.1，如果要计算与i1相似的其它商品，显然也只用计算i1与i2-i8的相似性，然后按照与i1的相似程度排序即可。

表4.3.2.1 用户的购买记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用户id  商品id | u1 | u2 | u3 | u4 | u5 |
| i1 | 1 | 1 | 1 | 1 | 1 |
| i2 | 1 | 1 | 1 | 1 | 0 |
| i3 | 1 | 1 | 1 | 0 | 0 |
| i4 | 1 | 1 | 0 | 0 | 0 |
| i5 | 1 | 0 | 0 | 0 | 0 |
| i6 |  |  |  |  |  |
| i7 |  |  |  |  |  |
| i8 |  |  |  |  |  |

## 7.5 推荐算法评价

推荐引擎作为一个工具—一个回答问题的工具。对于用户来说，什么样的推荐才是最好的？在探求答案之前最好还是探究下问题（什么样的推荐引擎最好）。到底什么样的推荐引擎才是最好的。用户如何知晓推荐引擎如何才能产生满足需要的答案？本章的余下部分将介绍如何评价推荐引擎，因为，在实现一个特殊的推荐系统时其评价标准相当重要。

最好的推荐引擎可能需要有一种“通灵”能力，在某种程度上，它可以在用户的行为发生之前，就对用户的某些偏好做出预测，预测某些用户对某一项目（item）的偏好。推荐系统可以预测将来你对所有项目的实际偏好。这样的推荐系统才可能是一种好的推荐系统。

事实上，大多数的推荐引擎实际上也就是对所有项目（item）的评分进行预测。所以评价推荐引擎的性能指标就是要评估其对项目的预测精度如何，即如果预测值和实际值之间的误差越小，则推荐系统的准确性越好。

### 7.5.1 训练数据和得分

虽然这些确切的评分偏好并不存在。也没有人知道（包括你自己）未来你对一个新的项目（item）评分是多少。所以，在推荐引擎中，用一部分真实的数据作为测试数据，来测试预测值和真实值之间的误差。在真实的测试数据中，需要去掉一部分已经被评分的项目偏好数据。然后将去除特定评分项的数据输入推荐系统，让推荐系统预测缺失项目的评分，这样就可以用预测值和真实值进行对比，就可以知道推荐系统的精度了。

按照以上的叙述，测试推荐引擎的性能就很简单了。比如，可以很方便的计算预测值和真实值之间的平均误差。平均误差越低越好，因为那意味着估计值更靠近真实值。如果误差为0.0，那说明估计值的结果是相当完美的—估计值和真实值之间没有误差。

有时候也使用均方根误差作为误差评价标准：即预测值和真实值差的平方和取平均值后再开方。依然是，这个值越小越好。表7.5.1.1展示了相关结果。

表7.5.1.1 预测值和真实值间的平均误差和均方根误差

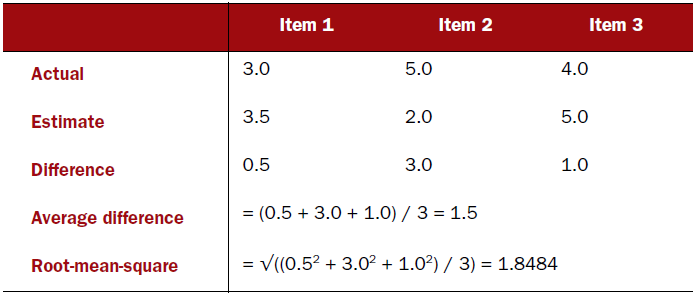


表7.5.1.1展示了一系列真实值和预测值之间的差异，从中可以看到如何计算平均误差和均方根误差的。均方误差计算方法对偏离程度大的项目（此处item2偏离度较大）有更大的惩罚（即误差越大，相比较而言均方误差的值比平均误差的值要大），在某些时候这种惩罚是符合实际需要的。For example, an estimate that’s off by two whole stars is probably more than twice as bad as one off by just one star。因为均方误差计算方法更直观，物理意义上也更好理解，所以后续的例子中将使用平均误差作为误差测量方法。

### 7.5.2 准确率和召回率

对用户偏好的估计并不仅仅是推荐系统要做的唯一的事，还有其它一些方面用来衡量推荐系统的好坏。一般来说推荐系统只要能够向用户推荐出一系列按照相关性从大到小顺序排列的结果就可以了，并不需要向用户显示预测的偏好是多少。事实上，大多数情况下推荐结果列表的顺序如何并无太大的关系，关键是结果列表中要有对用户有用的项目。

更一般的观点，可以将经典的信息检索度量方法：准确度和召回率，用来衡量推荐系统。这两个指标的典型应用时搜索引擎，搜索引擎返回与查询串相关的一系列可能的结果。

搜索引擎不应该返回与查询不相关的结果，但是其需要尽可能多的返回与查询相关的结果。精度是在搜索结果中与查询相关的结果与整个搜索结果的比例。精度等于10表明有在搜索结果中有10条结果与查询相关。召回率指的是查询结果中与查询相关的结果与整体相关结果的比例。图2.3展示了这两者的示意图。



图2.3 搜索引擎中准确率和召回率的关系

推荐系统的准确率指的是推荐结果中符合要求的结果与整个结果列表数量的比值，召回率指的是推荐结果中符合要求的结果（good recommendations）与整体符合要求的结果的比例。下节将介绍什么是符合要求的结果（good recommendations）。

# 八 技术前沿

## 1 深度学习

特征学习

## 2 流形学习

假设数据是均匀采样于一个高维欧氏空间中的低维流形，流形学习就是从高维采样数据中恢复低维流形结构，即找到高维空间中的低维流形，并求出相应的嵌入映射，以实现维数约简或者数据可视化。它是从观测到的现象中去寻找事物的本质，找到产生数据的内在规律

## 3 知识图谱

知识图谱(Mapping Knowledge Domain）也被称为科学知识图谱，在图书情报界称为知识域可视化或知识领域映射地图，是显示知识发展进程与结构关系的一系列各种不同的图形，用可视化技术描述知识资源及其载体，挖掘、分析、构建、绘制和显示知识及它们之间的相互联系。

具体来说，知识图谱是把应用数学、图形学、信息可视化技术、信息科学等学科的理论与方法与计量学引文分析、共现分析等方法结合，用可视化的图谱形象地展示学科的核心结构、发展历史、前沿领域以及整体知识架构的多学科融合的一种研究方法。它把复杂的知识领域通过数据挖掘、信息处理、知识计量和图形绘制而显示出来，揭示知识领域的动态发展规律，为学科研究提供切实的、有价值的参考。迄今为止，其实际应用在发达国家已经逐步拓展并取得了较好的效果，但它在我国仍属研究的起步阶段。

## 4 推荐阅读

《集体智慧编程》

《机器学习实战》

《统计学习方法》

《Pattern Recognition And Machine Learning》