机器学习实战

> 来源：apachecn/MachineLearning

> 欢迎任何人参与和完善：一个人可以走的很快，但是一群人却可以走的更远

- ApacheCN - 学习机器学习群【629470233】 
- Machine Learning in Action (　机器学习实战) | ApacheCN(apache 中文网)
- 电子版书籍：【机器学习实战-中文版-带目录版.pdf】
- 视频每周更新：如果你觉得有价值，请帮忙点 Star【后续组织学习活动：sklearn + tensorflow 】
- --- 视频网站：优酷 / bilibili / Acfun ，可直接在线播放。（最下方有相应链接）
- --- 对于帮忙转发的朋友，可以私聊 企鹅 赠送《机器学习实战》百度云视频一套，谢谢

第一部分 分类

- 1.) 机器学习基础
- 2.) k-近邻算法
- 3.) 决策树
- 4.) 基于概率论的分类方法：朴素贝叶斯
- 5.) Logistic回归
- 6.) 支持向量机
- 7.) 集成方法-随机森林和AdaBoost

第二部分 利用回归预测数值型数据

- 8.) 预测数值型数据：回归
- 9.) 树回归

第三部分 无监督学习

- 10.) 使用K-均值聚类算法对未标注数据分组:k-means聚类
- 11.) 使用Apriori算法进行关联分析
- 12.) 使用FP-growth算法来高效发现频繁项集

第四部分 其他工具

- 13.) 利用PCA来简化数据
- 14.) 利用SVD简化数据
- 15.) 大数据与MapReduce

第五部分 项目实战(非课本内容)

- 16.) 推荐系统

阶段性总结

- 2017-04-08_第一期的总结

联系方式

> 项目负责人

- @jiangzhonglian（片刻）

> 项目贡献者

- @jiangzhonglian（片刻）
- @wangyangting（那伊抹微笑）
- @chenyyx（瑶妹）
- @geekidentity（侯法超）
- @mikechengwei（Mike）
- @hello19883（hello19883）
- @sheepmen（徐鑫）
- @highfei2011（ibe）
- @LeeMoonCh（Arithmetic）

> 加入方式

- 企鹅: 529815144(片刻) 1042658081(那伊抹微笑) 190442212(瑶妹)

- ApacheCN(apache中文网) 维护更新

- 关于我们
- 加入我们

网站视频

> 知乎问答-爆炸啦-机器学习该怎么入门？

当然我知道，第一句就会被吐槽，因为科班出身的人，不屑的吐了一口唾沫，说傻X，还评论 Andrew Ng 的视频。。

我还知道还有一部分人，看 Andrew Ng 的视频就是看不懂，那神秘的数学推导，那迷之微笑的英文版的教学，我何尝又不是这样走过来的？？ 我的心可能比你们都痛，因为我在网上收藏过上10部《机器学习》相关视频，外加国内本土风格的教程：7月+小象 等等，我都很难去听懂，直到有一天，被一个百度的高级算法分析师推荐说：《机器学习实战》还不错，通俗易懂，你去试试？？

我试了试，还好我的Python基础和调试能力还不错，基本上代码都调试过一遍，很多高大上的 "理论+推导"，在我眼中变成了几个 "加减乘除+循环"，我想这不就是像我这样的程序员想要的入门教程么？

很多程序员说机器学习 TM 太难学了，是的，真 TM 难学，我想最难的是：没有一本像《机器学习实战》那样的作者愿意以程序员 Coding 角度去给大家讲解！！

最近几天，GitHub 涨了 300颗 star，加群的200人， 现在还在不断的增加++，我想大家可能都是感同身受吧！

很多想入门新手就是被忽悠着收藏收藏再收藏，但是最后还是什么都没有学到，也就是"资源收藏家"，也许新手要的就是 MachineLearning(机器学习) 学习路线图。没错，我可以给你们的一份，因为我们还通过视频记录下来我们的学习过程。水平当然也有限，不过对于新手入门，绝对没问题，如果你还不会，那算我输！！

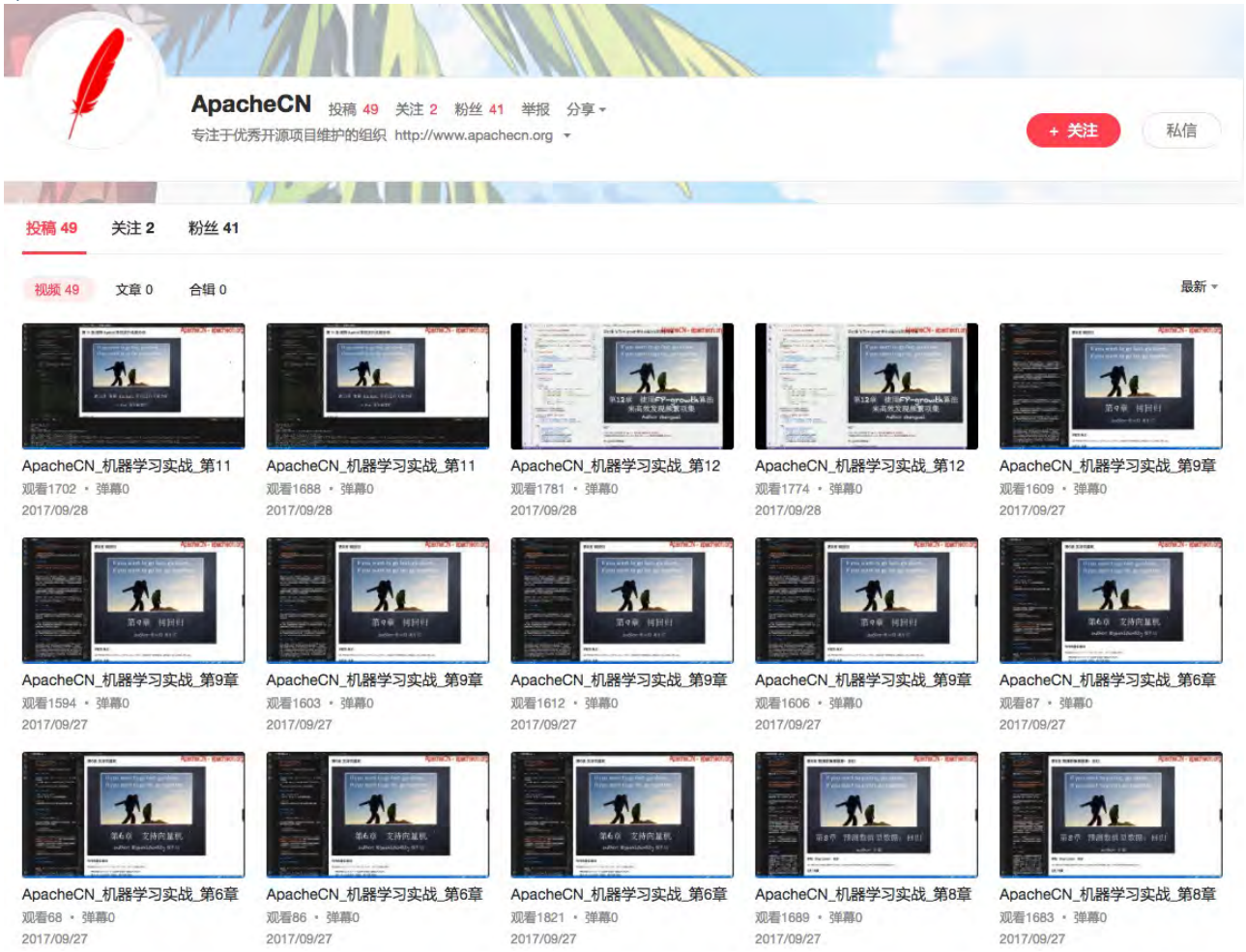
> 视频怎么看？

1. 理论科班出身-建议去学习 Andrew Ng 的视频（Ng 的视频绝对是权威，这个毋庸置疑）
2. 编码能力强 - 建议看我们的《机器学习实战-教学版》
3. 编码能力弱 - 建议看我们的《机器学习实战-讨论版》，不过在看理论的时候，看 教学版-理论部分；讨论版的废话太多，不过在讲解代码的时候是一行一行讲解的；所以，根据自己的需求，自由的组合。

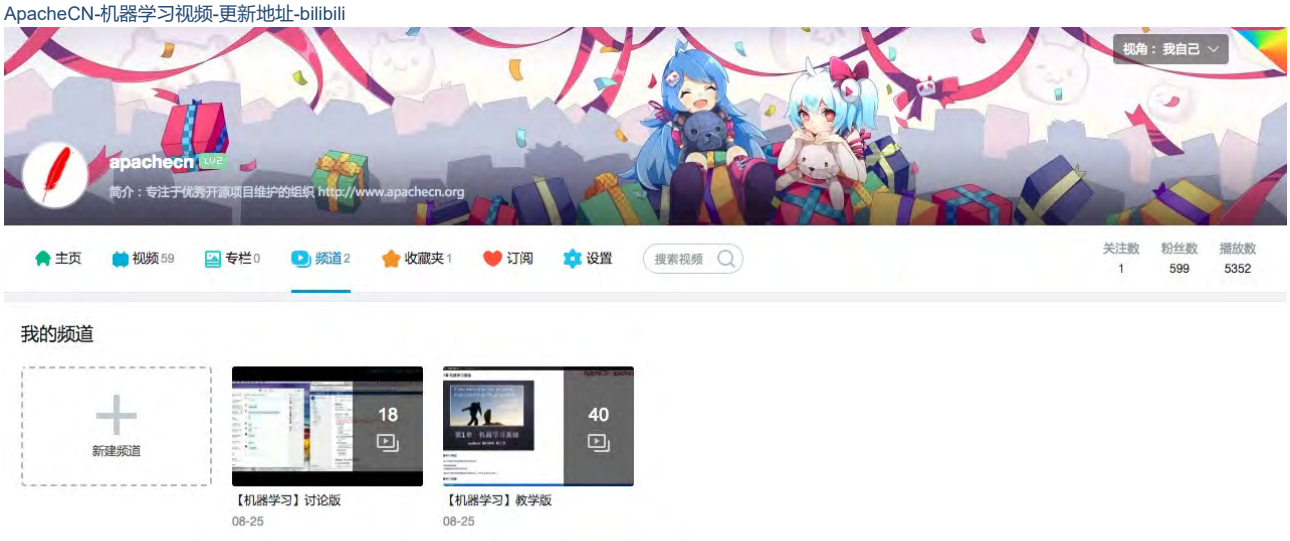
循序渐进大体介绍：机器学习初学者建议 | ApacheCN

干货内容实际操作：MachineLearning(机器学习) 学习路线图

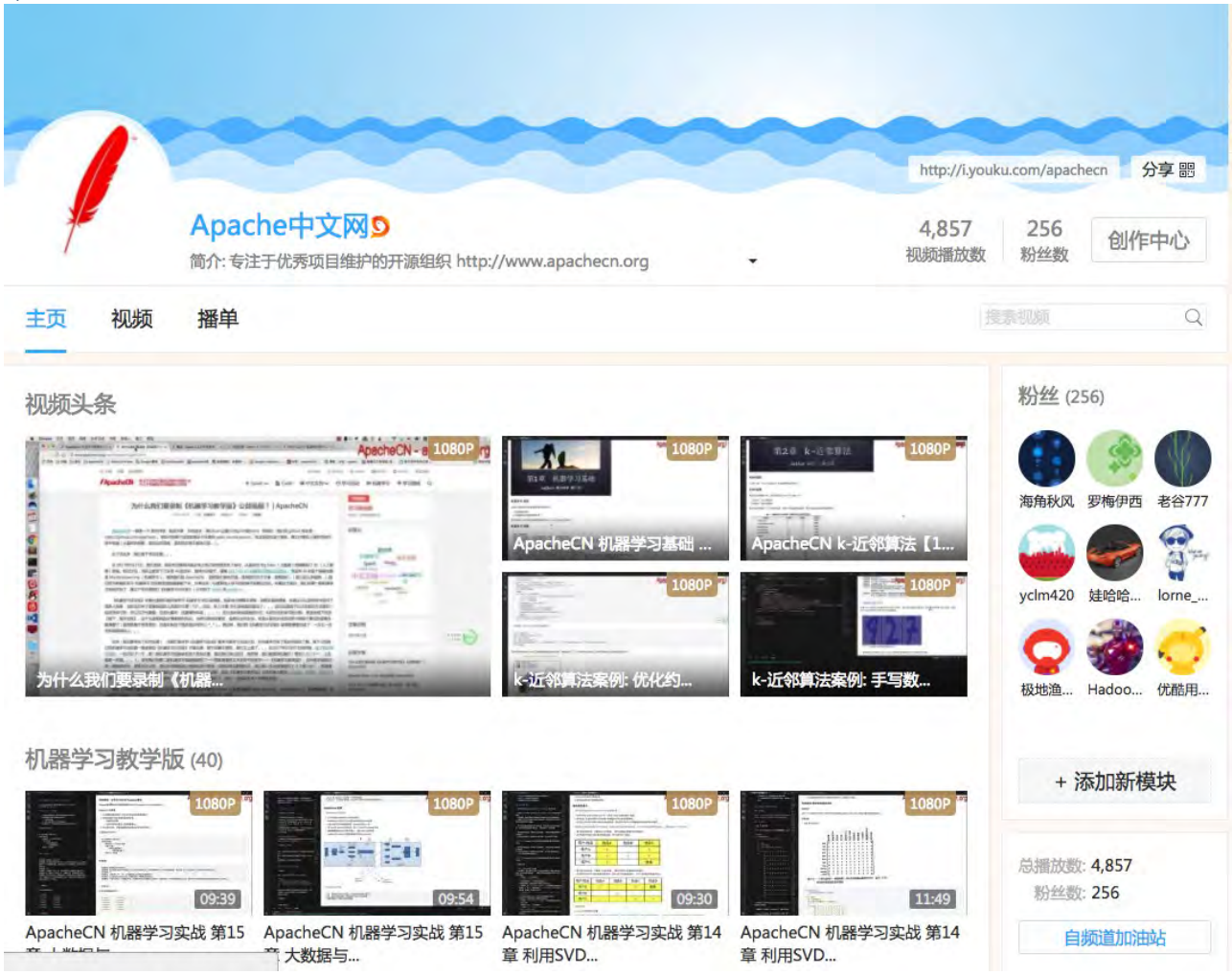
ApacheCN-机器学习视频-更新地址-AcFun



ApacheCN-机器学习视频-更新地址-bilibili



ApacheCN-机器学习视频-更新地址-优酷



其它中文文档

- Sklearn 0.19 中文文档
- Spark 2.2.0和2.0.2 中文文档
- Storm 1.1.0和1.0.1 中文文档
- Beam 中文文档
- TensorFlow R1.2 中文文档
- Kudu 1.4.0 中文文档
- Zeppelin 0.7.2 中文文档
- Elasticsearch 5.4 中文文档
- Kibana 5.2 中文文档

第1章 机器学习基础

机器学习 概述

`机器学习`就是把无序的数据转换成有用的信息。

1. 获取海量的数据
2. 从海量数据中获取有用的信息

我们会利用计算机来彰显数据背后的真实含义，这才是`机器学习`的意义。

机器学习 场景

```
例如：识别动物猫
模式识别（官方标准）：人们通过大量的经验，得到结论，从而判断它就是猫。
机器学习（数据学习）：人们通过阅读进行学习，观察它会叫、小眼睛、两只耳朵、四条腿、一条尾巴，得到结论，从而判断它就是猫。
深度学习（深入数据）：人们通过深入了解它，发现它会'喵喵'的叫、与同类的猫科动物很类似，得到结论，从而判断它就是猫。（深度学习常用领域：语音识别、图像识别）

模式识别（pattern recognition）：模式识别是最古老的（作为一个术语而言，可以说是很过时的）。
    我们把环境与客体统称为“模式”，识别是对模式的一种认知，是如何让一个计算机程序去做一些看起来很“智能”的事情。
    通过融于智慧和直觉后，通过构建程序，识别一些事物，而不是人，例如：识别数字。
机器学习（machine learning）：机器学习是最基础的（当下初创公司和研究实验室的热点领域之一）。
    在90年代初，人们开始意识到一种可以更有效地构建模式识别算法的方法，那就是用数据（可以通过廉价劳动力采集获得）去替换专家（具有很多图像方面知识的人）。
    “机器学习”强调的是，在给计算机程序（或者机器）输入一些数据后，它必须做一些事情，那就是学习这些数据，而这个学习的步骤是明确的。
    机器学习（Machine Learning）是一门专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身性能的学科。
深度学习（deep learning）：深度学习是非常崭新和有影响力的前沿领域，我们甚至不会去思考-后深度学习时代。
    深度学习是机器学习研究中的一个新的领域，其动机在于建立、模拟人脑进行分析学习的神经网络，它模仿人脑的机制来解释数据，例如图像，声音和文本。

参考地址：
http://www.csdn.net/article/2015-03-24/2824301
http://baike.baidu.com/link?url=76P-uA4EBrC3G-I__P1tqeO7eoDS709Kp4wYuHxc7GNkz_xn0NxuAtEohbpey7LUa2zUQLJxvIKUx4bnrEfOmsWLKbDmvG1PCoRkJisMTQka6-QR
```

> 机器学习已应用于多个领域，远远超出大多数人的想象，横跨：计算机科学、工程技术和统计学等多个学科。

- 搜索引擎: 根据你的搜索点击，优化你下次的搜索结果。
- 垃圾邮件: 会自动的过滤垃圾广告邮件到垃圾箱内。
- 超市优惠券: 你会发现，你在购买小孩子尿布的时候，售货员会赠送你一张优惠券可以兑换6罐啤酒。
- 邮局邮寄: 手写软件自动识别寄送贺卡的地址。
- 申请贷款: 通过你最近的金融活动信息进行综合评定，决定你是否合格。

机器学习 组成

主要任务

- 分类：将实例数据划分到合适的类别中。
- 回归：主要用于预测数值型数据。（ 示例：数据通过给定数据点来拟合最优曲线 ）

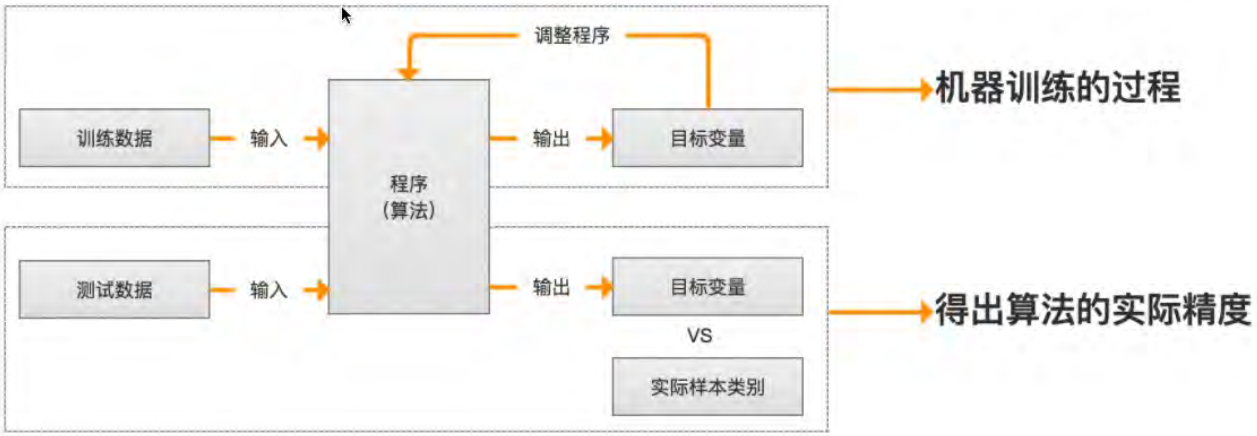
监督学习

- 必须确定目标变量的值，以便机器学习算法可以发现特征和目标变量之间的关系。 (包括：分类和回归)
- 样本集：训练数据 + 测试数据
  - 训练样本 = 特征(feature) + 目标变量(label: 分类-离散值/回归-连续值)
  - 特征通常是训练样本集的列，它们是独立测量得到的。
  - 目标变量: 目标变量是机器学习预测算法的测试结果。
    - 在分类算法中目标变量的类型通常是标称型(如：真与假)，而在回归算法中通常是连续型(如：1~100)。
- 知识表示：
  1. 可以采用规则集的形式【例如：数学成绩大于90分为优秀】
  2. 可以采用概率分布的形式【例如：通过统计分布发现，90%的同学数学成绩，在70分以下，那么大于70分定为优秀】
  3. 可以使用训练样本集中的一个实例【例如：通过样本集合，我们训练出一个模型实例，得出 年轻，数学成绩中高等，谈吐优雅，我们认为是优秀】

非监督学习

- 数据没有类别信息，也不会给定目标值。
- 聚类：在无监督学习中，将数据集分成由类似的对象组成多个类的过程称为聚类。
- 密度估计：将寻找描述数据统计值的过程称之为密度估计。【就是：根据训练样本确定x的概率分布】
- 此外，无监督学习还可以减少数据特征的维度，以便我们可以使用二维或三维图形更加直观地展示数据信息。

训练过程



算法汇总

表1-2 用于执行分类、回归、聚类和密度估计的机器学习算法

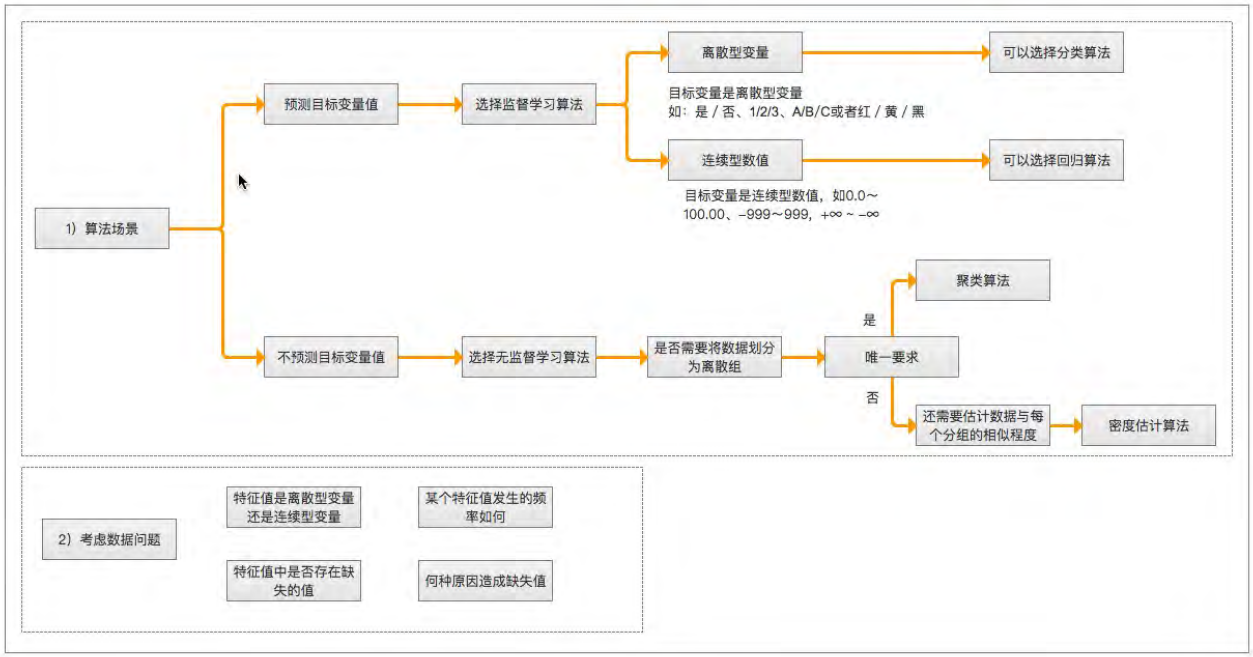
监督学习的用途	
k-近邻算法	线性回归
朴素贝叶斯算法	局部加权线性回归
支持向量机	Ridge 回归
决策树	Lasso 最小回归系数估计
无监督学习的用途	
K-均值	最大期望算法
DBSCAN	Parzen窗设计

机器学习 使用

> 选择算法需要考虑的两个问题

1. 算法场景
   - 预测明天是否下雨，因为可以用历史的天气情况做预测，所以选择监督学习算法
   - 给一群陌生的人进行分组，但是我们并没有这些人的类别信息，所以选择无监督学习算法、通过他们身高、体重等特征进行处理。
2. 需要收集或分析的数据是什么

> 举例



> 机器学习 开发流程

```
* 收集数据：收集样本数据
* 准备数据：注意数据的格式
* 分析数据：为了确保数据集中没有垃圾数据；
    如果是算法可以处理的数据格式或可信任的数据源，则可以跳过该步骤；
    另外该步骤需要人工干预，会降低自动化系统的价值。
* 训练算法：[机器学习算法核心]如果使用无监督学习算法，由于不存在目标变量值，则可以跳过该步骤
* 测试算法：[机器学习算法核心]评估算法效果
* 使用算法：将机器学习算法转为应用程序
```

Python 语言 优势

1. 可执行伪代码
2. Python比较流行：使用广泛、代码范例多、丰富模块库，开发周期短
3. Python语言的特色：清晰简练、易于理解
4. Python语言的缺点：唯一不足的是性能问题
5. Python相关的库
    - 科学函数库：`SciPy`、`NumPy`(底层语言：C和Fortran)
    - 绘图工具库：`Matplotlib`

第2章 k-近邻算法

KNN 概述

`k-近邻（kNN, k-NearestNeighbor）算法主要是用来进行分类的.`

KNN 场景

电影可以按照题材分类，那么如何区分 `动作片` 和 `爱情片` 呢？

1. 动作片：打斗次数更多
2. 爱情片：亲吻次数更多

基于电影中的亲吻、打斗出现的次数，使用 k-近邻算法构造程序，就可以自动划分电影的题材类型。

表2-1 每部电影的打斗镜头数、接吻镜头数以及电影评估类型

电影名称	打斗镜头	接吻镜头	电影类型
California Man	3	104	爱情片
He's Not Really into Dudes	2	100	爱情片
Beautiful Woman	1	81	爱情片
Kevin Longblade	101	10	动作片
Robo Slayer 3000	99	5	动作片
Amped II	98	2	动作片
?	18	90	未知

表2-2 已知电影与未知电影的距离

电影名称	与未知电影的距离
California Man	20.5
He's Not Really into Dudes	18.7
Beautiful Woman	19.2
Kevin Longblade	115.3
Robo Slayer 3000	117.4
Amped II	118.9

现在根据上面我们得到的样本集中所有电影与未知电影的距离，按照距离递增排序，可以找到 `k` 个距离最近的电影。
假定 `k=3`，则三个最靠近的电影依次是， `He's Not Really into Dudes` 、 `Beautiful Woman` 和 `California Man`。
`knn` 算法按照距离最近的三部电影的类型，决定未知电影的类型，而这三部电影全是爱情片，因此我们判定未知电影是爱情片。

KNN 原理

> KNN 工作原理

1. 假设有一个带有标签的样本数据集（训练样本集），其中包含每条数据与所属分类的对应关系。
2. 输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较。
    1. 计算新数据与样本数据集中每条数据的距离。
    2. 对求得的所有距离进行排序（从小到大，越小表示越相似）。
    3. 取前 k （k 一般小于等于 20 ）个样本数据对应的分类标签。
3. 求 k 个数据中出现次数最多的分类标签作为新数据的分类。

> KNN 开发流程

```
收集数据：任何方法
准备数据：距离计算所需要的数值，最好是结构化的数据格式
分析数据：任何方法
训练算法：此步骤不适用于 k-近邻算法
测试算法：计算错误率
使用算法：输入样本数据和结构化的输出结果，然后运行 k-近邻算法判断输入数据分类属于哪个分类，最后对计算出的分类执行后续处理
```

> KNN 算法特点

```
优点：精度高、对异常值不敏感、无数据输入假定
缺点：计算复杂度高、空间复杂度高
适用数据范围：数值型和标称型
```

KNN 项目案例

项目案例1: 优化约会网站的配对效果

项目概述

海伦使用约会网站寻找约会对象。经过一段时间之后，她发现曾交往过三种类型的人:

- 不喜欢的人
- 魅力一般的人
- 极具魅力的人

她希望：

1. 工作日与魅力一般的人约会
2. 周末与极具魅力的人约会
3. 不喜欢的人则直接排除掉

现在她收集到了一些约会网站未曾记录的数据信息，这更有助于匹配对象的归类。

开发流程

```
收集数据：提供文本文件
准备数据：使用 Python 解析文本文件
分析数据：使用 Matplotlib 画二维散点图
训练算法：此步骤不适用于 k-近邻算法
测试算法：使用海伦提供的部分数据作为测试样本。
        测试样本和非测试样本的区别在于：
            测试样本是已经完成分类的数据，如果预测分类与实际类别不同，则标记为一个错误。
使用算法：产生简单的命令行程序，然后海伦可以输入一些特征数据以判断对方是否为自己喜欢的类型。
```

> 收集数据：提供文本文件

海伦把这些约会对象的数据存放在文本文件 datingTestSet2.txt 中，总共有 1000 行。海伦约会的对象主要包含以下 3 种特征：

- 每年获得的飞行常客里程数
- 玩视频游戏所耗时间百分比
- 每周消费的冰淇淋公升数

文本文件数据格式如下：

```
40920   8.326976        0.953952        3
14488   7.153469        1.673904        2
26052   1.441871        0.805124        1
75136   13.147394       0.428964        1
38344   1.669788        0.134296        1
```

> 准备数据：使用 Python 解析文本文件

将文本记录转换为 NumPy 的解析程序

```
def file2matrix(filename):
    """
    Desc:
```

```
    导入训练数据
parameters:
    filename: 数据文件路径
return:
    数据矩阵 returnMat 和对应的类别 classLabelVector
"""
fr = open(filename)
# 获得文件中的数据行的行数
numberOfLines = len(fr.readlines())
# 生成对应的空矩阵
# 例如：zeros(2，3)就是生成一个 2*3的矩阵，各个位置上全是 0
returnMat = zeros((numberOfLines, 3))  # prepare matrix to return
classLabelVector = []  # prepare labels return
fr = open(filename)
index = 0
for line in fr.readlines():
    # str.strip([chars]) --返回移除字符串头尾指定的字符生成的新字符串
    line = line.strip()
    # 以 '\t' 切割字符串
    listFromLine = line.split('\t')
    # 每列的属性数据
    returnMat[index, :] = listFromLine[0:3]
    # 每列的类别数据，就是 label 标签数据
    classLabelVector.append(int(listFromLine[-1]))
    index += 1
# 返回数据矩阵returnMat和对应的类别classLabelVector
return returnMat, classLabelVector
```

> 分析数据：使用 Matplotlib 画二维散点图

```
import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(datingDataMat[:, 1], datingDataMat[:, 2], 15.0*array(datingLabels), 15.0*array(datingLabels))
plt.show()
```

下图中采用矩阵的第一和第三列属性得到很好的展示效果，清晰地标识了三个不同的样本分类区域，具有不同爱好的人其类别区域也不同。

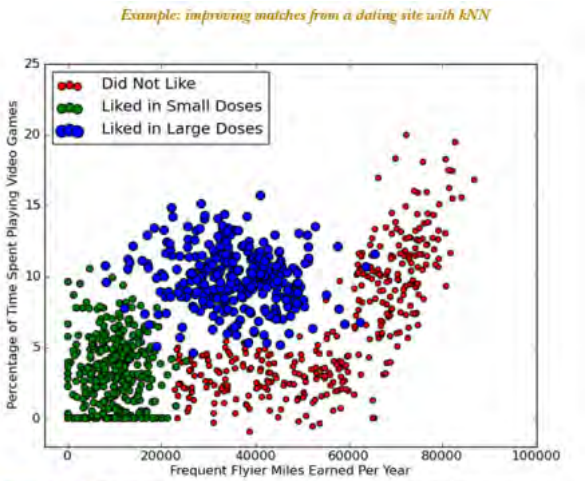


Figure 2.5 Dating data with frequent flier miles versus percentage of time spent playing video games plotted. The dating data has three features, and these two features show areas where the three different classes lie.

- 归一化数据（归一化是一个让权重变为统一的过程，更多细节请参考： https://www.zhihu.com/question/19951858 ）

序号	玩视频游戏所耗时间百分比	每年获得的飞行常客里程数	每周消费的冰淇淋公升数	样本分类
1	0.8	400	0.5	1
2	12	134 000	0.9	3
3	0	20 000	1.1	2
4	67	32 000	0.1	2

样本3和样本4的距离：

$$\sqrt{(0-67)^2+(20000-32000)^2+(1.1-0.1)^2}$$

归一化特征值，消除特征之间量级不同导致的影响

```
def autoNorm(dataSet):
    """
    Desc:
        归一化特征值，消除特征之间量级不同导致的影响
    parameter:
        dataSet: 数据集
    return:
        归一化后的数据集 normDataSet. ranges和minVals即最小值与范围，并没有用到

    归一化公式：
        Y = (X-Xmin)/(Xmax-Xmin)
        其中的 min 和 max 分别是数据集中的最小特征值和最大特征值。该函数可以自动将数字特征值转化为0到1的区间。
    """
    # 计算每种属性的最大值、最小值、范围
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    # 极差
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    # 生成与最小值之差组成的矩阵
    normDataSet = dataSet - tile(minVals, (m, 1))
    # 将最小值之差除以范围组成矩阵
    normDataSet = normDataSet / tile(ranges, (m, 1))  # element wise divide
    return normDataSet, ranges, minVals
```

> 训练算法：此步骤不适用于 k-近邻算法

因为测试数据每一次都要与全量的训练数据进行比较，所以这个过程是没有必要的。

> 测试算法：使用海伦提供的部分数据作为测试样本。如果预测分类与实际类别不同，则标记为一个错误。

kNN 分类器针对约会网站的测试代码

```
def datingClassTest():
    """
    Desc:
        对约会网站的测试方法
    parameters:
        none
    return:
        错误数
    """
    # 设置测试数据的的一个比例（训练数据集比例=1-hoRatio）
    hoRatio = 0.1  # 测试范围,一部分测试一部分作为样本
    # 从文件中加载数据
    datingDataMat, datingLabels = file2matrix('input/2.KNN/datingTestSet2.txt')  # load data setfrom file
    # 归一化数据
    normMat, ranges, minVals = autoNorm(datingDataMat)
    # m 表示数据的行数，即矩阵的第一维
    m = normMat.shape[0]
    # 设置测试的样本数量， numTestVecs:m表示训练样本的数量
    numTestVecs = int(m * hoRatio)
    print 'numTestVecs=', numTestVecs
    errorCount = 0.0
    for i in range(numTestVecs):
        # 对数据测试
        classifierResult = classify0(normMat[i, :], normMat[numTestVecs:m, :], datingLabels[numTestVecs:m], 3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierResult, datingLabels[i])
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print "the total error rate is: %f" % (errorCount / float(numTestVecs))
    print errorCount
```

> 使用算法：产生简单的命令行程序，然后海伦可以输入一些特征数据以判断对方是否为自己喜欢的类型。

约会网站预测函数

```
def clasdifyPerson():
    resultList = ['not at all', 'in small doses', 'in large doses']
    percentTats = float(raw_input("percentage of time spent playing video games ?"))
    ffMiles = float(raw_input("frequent filer miles earned per year?"))
    iceCream = float(raw_input("liters of ice cream consumed per year?"))
    datingDataMat, datingLabels = file2matrix('datingTestSet2.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)
    inArr = array([ffMils, percentTats, iceCream])
    classifierResult = classify0((inArr-minVals)/ranges,normMat,datingLabels, 3)
    print "You will probably like this person: ", resultList[classifierResult - 1]
```

实际运行效果如下:

```
>>> kNN.classifyPerson()
percentage of time spent playing video games?10
frequent flier miles earned per year?10000
liters of ice cream consumed per year?0.5
You will probably like this person: in small doses
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/2.KNN/kNN.py

项目案例2: 手写数字识别系统

项目概述

构造一个能识别数字 0 到 9 的基于 KNN 分类器的手写数字识别系统。

需要识别的数字是存储在文本文件中的具有相同的色彩和大小：宽高是 32 像素 * 32 像素的黑白图像。

开发流程

```
收集数据：提供文本文件。
准备数据：编写函数 img2vector()，将图像格式转换为分类器使用的向量格式
分析数据：在 Python 命令提示符中检查数据，确保它符合要求
训练算法：此步骤不适用于 KNN
测试算法：编写函数使用提供的部分数据集作为测试样本，测试样本与非测试样本的
          区别在于测试样本是已经完成分类的数据，如果预测分类与实际类别不同，
          则标记为一个错误
使用算法：本例没有完成此步骤，若你感兴趣可以构建完整的应用程序，从图像中提取
          数字，并完成数字识别，美国的邮件分拣系统就是一个实际运行的类似系统
```

> 收集数据: 提供文本文件

目录 trainingDigits 中包含了大约 2000 个例子，每个例子内容如下图所示，每个数字大约有 200 个样本；目录 testDigits 中包含了大约 900 个测试数据。

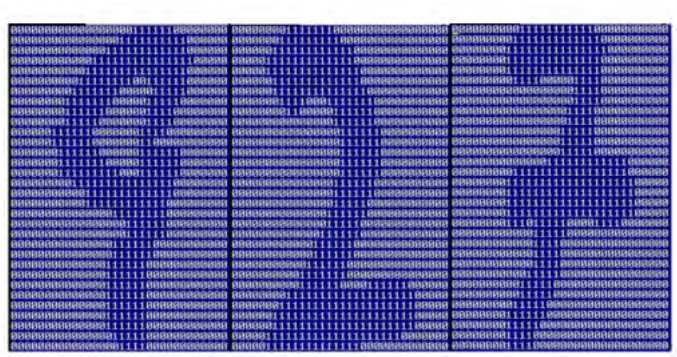


Figure 2.6 Examples of the handwritten digits dataset

> 准备数据: 编写函数 img2vector(), 将图像文本数据转换为分类器使用的向量

将图像文本数据转换为向量

```
def img2vector(filename):
    returnVect = zeros((1,1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readLine()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect
```

> 分析数据：在 Python 命令提示符中检查数据，确保它符合要求

在 Python 命令行中输入下列命令测试 img2vector 函数，然后与文本编辑器打开的文件进行比较:

```
>>> testVector = kNN.img2vector('testDigits/0_13.txt')
>>> testVector[0,0:31]
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
>>> testVector[0,31:63]
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

> 训练算法：此步骤不适用于 KNN

因为测试数据每一次都要与全量的训练数据进行比较，所以这个过程是没有必要的。

> 测试算法：编写函数使用提供的部分数据集作为测试样本，如果预测分类与实际类别不同，则标记为一个错误

```
def handwritingClassTest():
    # 1. 导入训练数据
    hwLabels = []
```

```
    trainingFileList = listdir('input/2.KNN/trainingDigits')  # load the training set
    m = len(trainingFileList)
    trainingMat = zeros((m, 1024))
    # hwLabels存储0～9对应的index位置， trainingMat存放的每个位置对应的图片向量
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]  # take off .txt
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        # 将 32*32的矩阵->1*1024的矩阵
        trainingMat[i, :] = img2vector('input/2.KNN/trainingDigits/%s' % fileNameStr)

    # 2. 导入测试数据
    testFileList = listdir('input/2.KNN/testDigits')  # iterate through the test set
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]  # take off .txt
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('input/2.KNN/testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierResult, classNumStr)
        if (classifierResult != classNumStr): errorCount += 1.0
    print "\nthe total number of errors is: %d" % errorCount
    print "\nthe total error rate is: %f" % (errorCount / float(mTest))
```

> 使用算法：本例没有完成此步骤，若你感兴趣可以构建完整的应用程序，从图像中提取数字，并完成数字识别，美国的邮件分拣系统就是一个实际运行的类似系统

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/2.KNN/kNN.py

---

- 作者：羊三 小瑶
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第3章 决策树

决策树 概述

决策树（Decision Tree）算法主要用来处理分类问题，是最经常使用的数据挖掘算法之一。

决策树 场景

一个叫做 "二十个问题" 的游戏，游戏的规则很简单：参与游戏的一方在脑海中想某个事物，其他参与者向他提问，只允许提 20 个问题，问题的答案也只能用对或错回答。问问题的人通过推断分解，逐步缩小待猜测事物的范围，最后得到游戏的答案。

一个邮件分类系统，大致工作流程如下：

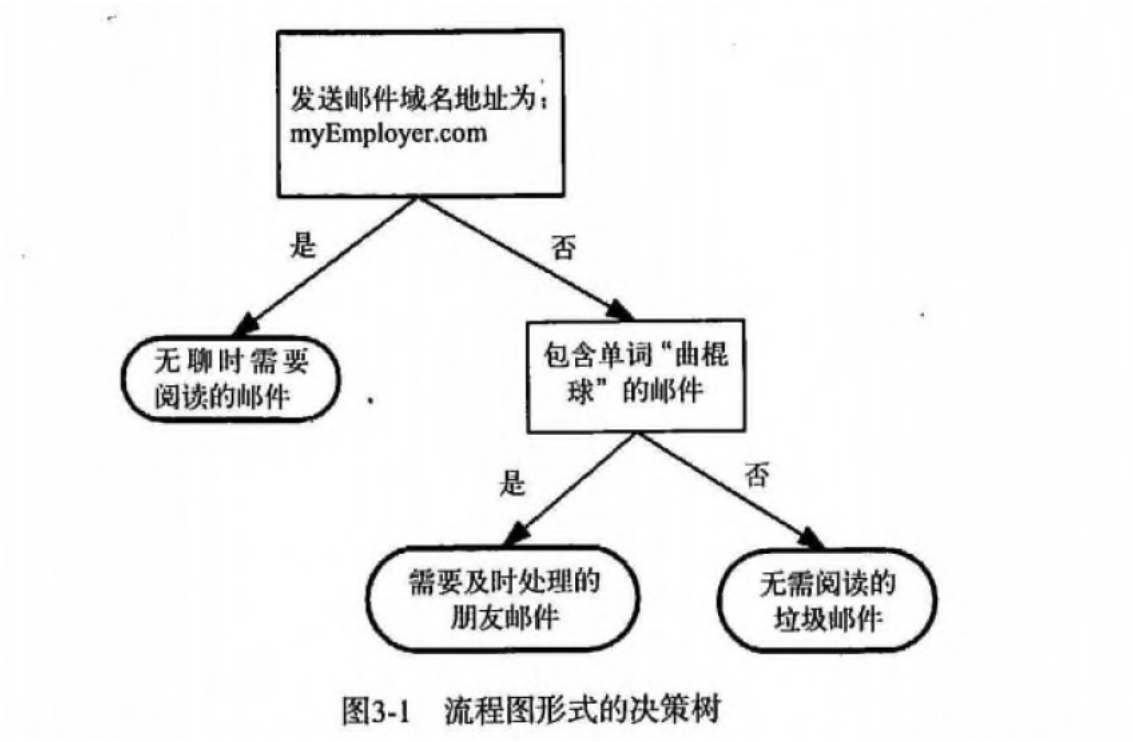


图3-1　流程图形式的决策树

```
首先检测发送邮件域名地址。如果地址为 myEmployer.com, 则将其放在分类 "无聊时需要阅读的邮件"中。
如果邮件不是来自这个域名，则检测邮件内容里是否包含单词 "曲棍球" , 如果包含则将邮件归类到 "需要及时处理的朋友邮件",
如果不包含则将邮件归类到 "无需阅读的垃圾邮件" 。
```

决策树 原理

决策树 须知概念

信息熵 & 信息增益

熵： 熵（entropy）指的是体系的混乱的程度，在不同的学科中也有引申出的更为具体的定义，是各领域十分重要的参量。

信息熵（香农熵）： 是一种信息的度量方式，表示信息的混乱程度，也就是说：信息越有序，信息熵越低。例如：火柴有序放在火柴盒里，熵值很低，相反，熵值很高。

信息增益： 在划分数据集前后信息发生的变化称为信息增益。

决策树 工作原理

如何构造一个决策树?
我们使用 createBranch() 方法，如下所示：

```
检测数据集中的所有数据的分类标签是否相同:
    If so return 类标签
    Else:
        寻找划分数据集的最好特征（划分之后信息熵最小，也就是信息增益最大的特征）
        划分数据集
        创建分支节点
            for 每个划分的子集
                调用函数 createBranch （创建分支的函数）并增加返回结果到分支节点中
        return 分支节点
```

决策树 开发流程

```
收集数据：可以使用任何方法。
准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化。
分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期。
训练算法：构造树的数据结构。
测试算法：使用经验树计算错误率。（经验树没有搜索到较好的资料，有兴趣的同学可以来补充）
使用算法：此步骤可以适用于任何监督学习算法，而使用决策树可以更好地理解数据的内在含义。
```

决策树 算法特点

```
优点：计算复杂度不高，输出结果易于理解，对中间值的缺失不敏感，可以处理不相关特征数据。
缺点：可能会产生过度匹配问题。
适用数据类型：数值型和标称型。
```

决策树 项目案例

项目案例1: 判定鱼类和非鱼类

项目概述

根据以下 2 个特征，将动物分成两类：鱼类和非鱼类。

特征：

1. 不浮出水面是否可以生存
2. 是否有脚蹼

开发流程

```
收集数据：可以使用任何方法
准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化
分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期
训练算法：构造树的数据结构
测试算法：使用决策树执行分类
使用算法：此步骤可以适用于任何监督学习算法，而使用决策树可以更好地理解数据的内在含义
```

> 收集数据：可以使用任何方法

	Can survive without coming to surface?	Has flippers?	Fish?
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	No	No
4	No	Yes	No
5	No	Yes	No

我们利用 createDataSet() 函数输入数据

```
def createDataSet():
    dataSet = [[1, 1, 'yes'],
            [1, 1, 'yes'],
            [1, 0, 'no'],
            [0, 1, 'no'],
            [0, 1, 'no']]
    labels = ['no surfacing', 'flippers']
    return dataSet, labels
```

> 准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化

此处，由于我们输入的数据本身就是离散化数据，所以这一步就省略了。

> 分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期

其中$p(x_i)$是选择该分类的概率。

为了计算熵，我们需要计算所有类别所有可能值包含的信息期望值，通过下面的公式得到：

$$H = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

计算给定数据集的香农熵的函数

```
def calcShannonEnt(dataSet):
    # 求list的长度，表示计算参与训练的数据量
    numEntries = len(dataSet)
    # 计算分类标签label出现的次数
    labelCounts = {}
    # the the number of unique elements and their occurance
    for featVec in dataSet:
        # 将当前实例的标签存储，即每一行数据的最后一个数据代表的是标签
        currentLabel = featVec[-1]
        # 为所有可能的分类创建字典，如果当前的键值不存在，则扩展字典并将当前键值加入字典。每个键值都记录了当前类别出现的次数。
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1

    # 对于 label 标签的占比，求出 label 标签的香农熵
```

```
    shannonEnt = 0.0
    for key in labelCounts:
        # 使用所有类标签的发生频率计算类别出现的概率。
        prob = float(labelCounts[key])/numEntries
        # 计算香农熵，以 2 为底求对数
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt
```

按照给定特征划分数据集

将指定特征的特征值等于 value 的行剩下列作为子数据集。

```
def splitDataSet(dataSet, index, value):
    """splitDataSet(通过遍历dataSet数据集，求出index对应的colnum列的值为value的行)
        就是依据index列进行分类，如果index列的数据等于 value的时候，就要将 index 划分到我们创建的新的数据集中
    Args:
        dataSet 数据集                  待划分的数据集
        index 表示每一行的index列        划分数据集的特征
        value 表示index列对应的value值   需要返回的特征的值。
    Returns:
        index列为value的数据集【该数据集需要排除index列】
    """
    retDataSet = []
    for featVec in dataSet:
        # index列为value的数据集【该数据集需要排除index列】
        # 判断index列的值是否为value
        if featVec[index] == value:
            # chop out index used for splitting
            # [:index]表示前index行，即若 index 为2，就是取 featVec 的前 index 行
            reducedFeatVec = featVec[:index]
            '''
            请百度查询一下： extend和append的区别
            list.append(object) 向列表中添加一个对象object
            list.extend(sequence) 把一个序列seq的内容添加到列表中
            1、使用append的时候，是将new_media看作一个对象，整体打包添加到music_media对象中。
            2、使用extend的时候，是将new_media看作一个序列，将这个序列和music_media序列合并，并放在其后面。
            result = []
            result.extend([1,2,3])
            print result
            result.append([4,5,6])
            print result
            result.extend([7,8,9])
            print result
            结果：
            [1, 2, 3]
            [1, 2, 3, [4, 5, 6]]
            [1, 2, 3, [4, 5, 6], 7, 8, 9]
            '''
            reducedFeatVec.extend(featVec[index+1:])
            # [index+1:]表示从跳过 index 的 index+1行，取接下来的数据
            # 收集结果值 index列为value的行【该行需要排除index列】
            retDataSet.append(reducedFeatVec)
    return retDataSet
```

选择最好的数据集划分方式

```
def chooseBestFeatureToSplit(dataSet):
    """chooseBestFeatureToSplit(选择最好的特征)

    Args:
        dataSet 数据集
    Returns:
        bestFeature 最优的特征列
    """
    # 求第一行有多少列的 Feature, 最后一列是label列嘛
    numFeatures = len(dataSet[0]) - 1
    # 数据集的原始信息熵
    baseEntropy = calcShannonEnt(dataSet)
    # 最优的信息增益值, 和最优的Featurn编号
    bestInfoGain, bestFeature = 0.0, -1
    # iterate over all the features
    for i in range(numFeatures):
        # create a list of all the examples of this feature
        # 获取对应的feature下的所有数据
        featList = [example[i] for example in dataSet]
        # get a set of unique values
        # 获取剔重后的集合，使用set对list数据进行去重
        uniqueVals = set(featList)
        # 创建一个临时的信息熵
        newEntropy = 0.0
        # 遍历某一列的value集合，计算该列的信息熵
```

```
        # 遍历当前特征中的所有唯一属性值，对每个唯一属性值划分一次数据集，计算数据集的新熵值，并对所有唯一特征值得到的熵求和。
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            # 计算概率
            prob = len(subDataSet)/float(len(dataSet))
            # 计算信息熵
            newEntropy += prob * calcShannonEnt(subDataSet)
        # gain[信息增益]: 划分数据集前后的信息变化， 获取信息熵最大的值
        # 信息增益是熵的减少或者是数据无序度的减少。最后，比较所有特征中的信息增益，返回最好特征划分的索引值。
        infoGain = baseEntropy - newEntropy
        print 'infoGain=', infoGain, 'bestFeature=', i, baseEntropy, newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature
```

问：上面的 newEntropy 为什么是根据子集计算的呢？
答：因为我们在根据一个特征计算香农熵的时候，该特征的分类值是相同，这个特征这个分类的香农熵为 0；
这就是为什么计算新的香农熵的时候使用的是子集。

> 训练算法：构造树的数据结构

创建树的函数代码如下：

```
def createTree(dataSet, labels):
    classList = [example[-1] for example in dataSet]
    # 如果数据集的最后一列的第一个值出现的次数=整个集合的数量，也就说只有一个类别，就只直接返回结果就行
    # 第一个停止条件：所有的类标签完全相同，则直接返回该类标签。
    # count() 函数是统计括号中的值在list中出现的次数
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    # 如果数据集只有1列，那么最初出现label次数最多的一类，作为结果
    # 第二个停止条件：使用完了所有特征，仍然不能将数据集划分成仅包含唯一类别的分组。
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)

    # 选择最优的列，得到最优列对应的label含义
    bestFeat = chooseBestFeatureToSplit(dataSet)
    # 获取label的名称
    bestFeatLabel = labels[bestFeat]
    # 初始化myTree
    myTree = {bestFeatLabel: {}}
    # 注：labels列表是可变对象，在PYTHON函数中作为参数时传址引用，能够被全局修改
    # 所以这行代码导致函数外的同名变量被删除了元素，造成例句无法执行，提示'no surfacing' is not in list
    del(labels[bestFeat])
    # 取出最优列，然后它的branch做分类
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        # 求出剩余的标签label
        subLabels = labels[:]
        # 遍历当前选择特征包含的所有属性值，在每个数据集划分上递归调用函数createTree()
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)
        # print 'myTree', value, myTree
    return myTree
```

> 测试算法：使用决策树执行分类

```
def classify(inputTree, featLabels, testVec):
    """classify(给输入的节点，进行分类)

    Args:
        inputTree  决策树模型
        featLabels Feature标签对应的名称
        testVec    测试输入的数据
    Returns:
        classLabel 分类的结果值，需要映射label才能知道名称
    """
    # 获取tree的根节点对于的key值
    firstStr = inputTree.keys()[0]
    # 通过key得到根节点对应的value
    secondDict = inputTree[firstStr]
    # 判断根节点名称获取根节点在label中的先后顺序，这样就知道输入的testVec怎么开始对照树来做分类
    featIndex = featLabels.index(firstStr)
    # 测试数据，找到根节点对应的label位置，也就知道从输入的数据的第几位来开始分类
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    print '+++', firstStr, 'xxx', secondDict, '---', key, '>>>', valueOfFeat
    # 判断分枝是否结束: 判断valueOfFeat是否是dict类型
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
```

```
    else:
        classLabel = valueOfFeat
    return classLabel
```

> 使用算法：此步骤可以适用于任何监督学习算法，而使用决策树可以更好地理解数据的内在含义。

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/3.DecisionTree/DecisionTree.py

项目案例2: 使用决策树预测隐形眼镜类型

项目概述

隐形眼镜类型包括硬材质、软材质以及不适合佩戴隐形眼镜。我们需要使用决策树预测患者需要佩戴的隐形眼镜类型。

开发流程

1. 收集数据: 提供的文本文件。
2. 解析数据: 解析 tab 键分隔的数据行
3. 分析数据: 快速检查数据，确保正确地解析数据内容，使用 createPlot() 函数绘制最终的树形图。
4. 训练算法: 使用 createTree() 函数。
5. 测试算法: 编写测试函数验证决策树可以正确分类给定的数据实例。
6. 使用算法: 存储树的数据结构，以便下次使用时无需重新构造树。

> 收集数据：提供的文本文件

文本文件数据格式如下：

```
young    myope    no      reduced no lenses
pre      myope    no      reduced no lenses
presbyopic       myope    no      reduced no lenses
```

> 解析数据：解析 tab 键分隔的数据行

```
lecses = [inst.strip().split('\t') for inst in fr.readlines()]
lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
```

> 分析数据：快速检查数据，确保正确地解析数据内容，使用 createPlot() 函数绘制最终的树形图。

```
>>> treePlotter.createPlot(lensesTree)
```

> 训练算法：使用 createTree() 函数

```
>>> lensesTree = trees.createTree(lenses, lensesLabels)
>>> lensesTree
{'tearRate': {'reduced': 'no lenses', 'normal': {'astigmatic':{'yes':
{'prescript':{'hyper':{'age':{'pre':'no lenses', 'presbyopic':
'no lenses', 'young':'hard'}}, 'myope':'hard'}}, 'no':{'age':{'pre':
'soft', 'presbyopic':{'prescript': {'hyper':'soft', 'myope':
'no lenses'}}, 'young':'soft'}}}}}
```

> 测试算法: 编写测试函数验证决策树可以正确分类给定的数据实例。

> 使用算法: 存储树的数据结构，以便下次使用时无需重新构造树。

使用 pickle 模块存储决策树

```
def storeTree(inputTree, filename):
    impory pickle
    fw = open(filename, 'w')
    pickle.dump(inputTree, fw)
    fw.close()

def grabTree(filename):
    import pickle
    fr = open(filename)
    return pickle.load(fr)
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/3.DecisionTree/DecisionTree.py

---

- 作者：片刻 小瑶
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第4章 基于概率论的分类方法：朴素贝叶斯

朴素贝叶斯 概述

贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。本章首先介绍贝叶斯分类算法的基础—贝叶斯定理。最后，我们通过实例来讨论贝叶斯分类的中最简

贝叶斯理论 & 条件概率

贝叶斯理论

我们现在有一个数据集，它由两类数据组成，数据分布如下图所示：

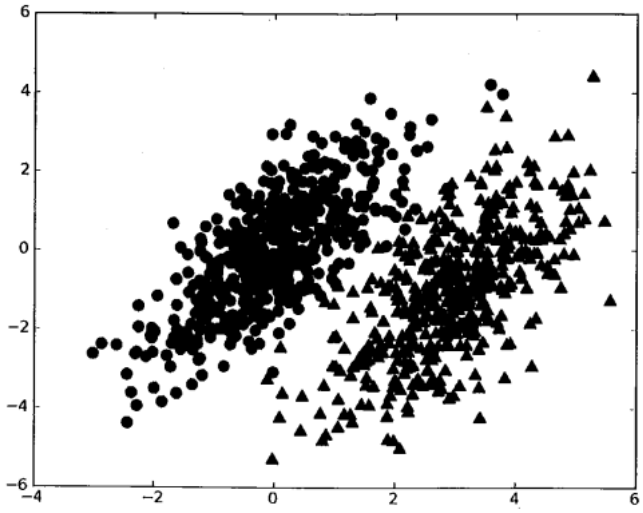


图4-1 两个参数已知的概率分布，参数决定了分布的形状

我们现在用 p1(x,y) 表示数据点 (x,y) 属于类别 1（图中用圆点表示的类别）的概率，用 p2(x,y) 表示数据点 (x,y) 属于类别 2（图中三角形表示的类别）的概率，那么对于一个新数据点 (x,y)，可以用下面的规则来判断它的类别：

- 如果 p1(x,y) > p2(x,y) ，那么类别为1
- 如果 p2(x,y) > p1(x,y) ，那么类别为2

也就是说，我们会选择高概率对应的类别。这就是贝叶斯决策理论的核心思想，即选择具有最高概率的决策。

条件概率

如果你对 p(x,y|c1) 符号很熟悉，那么可以跳过本小节。

有一个装了 7 块石头的罐子，其中 3 块是白色的，4 块是黑色的。如果从罐子中随机取出一块石头，那么是白色石头的可能性是多少？由于取石头有 7 种可能，其中 3 种为白色，所以取出白色石头的概率为 3/7 。那么取到黑色石头的概率又是多少呢？很显然，是 4/7 。我们使用 P(white) 来表示取到白色石头的概率，其概率值可以通过白色石头数目除以总的石头数目来得到。

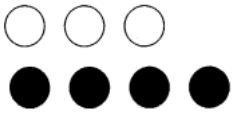


图4-2 一个包含7块石头的集合，石头的颜色为灰色或者黑色。如果随机从中取一块石头，那么取到灰色石头的概率为3/7。类似地，取到黑色石头的概率为4/7

如果这 7 块石头如下图所示，放在两个桶中，那么上述概率应该如何计算？

图4-3 落到两个桶中的7块石头

计算 P(white) 或者 P(black) ，如果事先我们知道石头所在桶的信息是会改变结果的。这就是所谓的条件概率（conditional probablity）。假定计算的是从 B 桶取到白色石头的概率，这个概率可以记作 P(white|bucketB) ，我们称之为“在已知石头出自 B 桶的条件下，取出白色石头的概率”。很容易得到，P(white|bucketA) 值为 2/4 ，P(white|bucketB) 的值为 1/3 。

条件概率的计算公式如下：

P(white|bucketB) = P(white and bucketB) / P(bucketB)

首先，我们用 B 桶中白色石头的个数除以两个桶中总的石头数，得到 P(white and bucketB) = 1/7 .其次，由于 B 桶中有 3 块石头，而总石头数为 7 ，于是 P(bucketB) 就等于 3/7 。于是又 P(white|bucketB) = P(white and bucketB) / P(bucketB) = (1/7) / (3/7) = 1/3 。

另外一种有效计算条件概率的方法称为贝叶斯准则。贝叶斯准则告诉我们如何交换条件概率中的条件与结果，即如果已知 P(x|c)，要求 P(c|x)，那么可以使用下面的计算方法：

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

使用条件概率来分类

上面我们提到贝叶斯决策理论要求计算两个概率 p1(x, y) 和 p2(x, y):

- 如果 p1(x, y) > p2(x, y), 那么属于类别 1;
- 如果 p2(x, y) > p1(X, y), 那么属于类别 2.

这并不是贝叶斯决策理论的所有内容。使用 p1() 和 p2() 只是为了尽可能简化描述，而真正需要计算和比较的是 p(c1|x, y) 和 p(c2|x, y) .这些符号所代表的具体意义是: 给定某个由 x、y 表示的数据点，那么该数据点来自类别 c1 的概率是多少？数据点来自类别 c2 的概率又是多少？注意这些概率与概率 p(x, y|c1) 并不一样，不过可以使用贝叶斯准则来交换概率中条件与结果。具体地，应用贝叶斯准则得到:

$$p(c_i|x,y) = \frac{p(x,y|c_i)p(c_i)}{p(x,y)}$$

使用上面这些定义，可以定义贝叶斯分类准则为:

- 如果 P(c1|x, y) > P(c2|x, y), 那么属于类别 c1;
- 如果 P(c2|x, y) > P(c1|x, y), 那么属于类别 c2.

在文档分类中，整个文档（如一封电子邮件）是实例，而电子邮件中的某些元素则构成特征。我们可以观察文档中出现的词，并把每个词作为一个特征，而每个词的出现或者不出现作为该特征的值，这样得到的特征数目就会跟词汇表中的词的数目一样多。

我们假设特征之间 相互独立 。所谓 独立(independence) 指的是统计意义上的独立，即一个特征或者单词出现的可能性与它和其他单词相邻没有关系，比如说，“我们”中的“我”和“们”出现的概率与这两个字相邻没有任何关系。这个假设正是朴素贝叶斯分类器中 朴素(naive) 一词的含义。朴素贝叶斯分类器中的另一个假设是，每个特征同等重要。

Note: 朴素贝叶斯分类器通常有两种实现方式: 一种基于伯努利模型实现，一种基于多项式模型实现。这里采用前一种实现方式。该实现方式中并不考虑词在文档中出现的次数，只考虑出不出现，因此在这个意义上相当于假设词是等权重的。

朴素贝叶斯 场景

机器学习的一个重要应用就是文档的自动分类。

在文档分类中，整个文档（如一封电子邮件）是实例，而电子邮件中的某些元素则构成特征。我们可以观察文档中出现的词，并把每个词作为一个特征，而每个词的出现或者不出现作为该特征的值，这样得到的特征数目就会跟词汇表中的词的数目一样多。

朴素贝叶斯是上面介绍的贝叶斯分类器的一个扩展，是用于文档分类的常用算法。下面我们会进行一些朴素贝叶斯分类的实践项目。

朴素贝叶斯 原理

朴素贝叶斯 工作原理

```
提取所有文档中的词条并进行去重
获取文档的所有类别
计算每个类别中的文档数目
对每篇训练文档:
    对每个类别:
        如果词条出现在文档中-->增加该词条的计数值（for循环或者矩阵相加）
        增加所有词条的计数值（此类别下词条总数）
对每个类别:
    对每个词条:
        将该词条的数目除以总词条数目得到的条件概率（P(词条|类别)）
返回该文档属于每个类别的条件概率（P(类别|文档的所有词条)）
```

朴素贝叶斯 开发流程

```
收集数据: 可以使用任何方法。
准备数据: 需要数值型或者布尔型数据。
分析数据: 有大量特征时，绘制特征作用不大，此时使用直方图效果更好。
训练算法: 计算不同的独立特征的条件概率。
测试算法: 计算错误率。
使用算法: 一个常见的朴素贝叶斯应用是文档分类。可以在任意的分类场景中使用朴素贝叶斯分类器，不一定非要是文本。
```

朴素贝叶斯 算法特点

```
优点: 在数据较少的情况下仍然有效，可以处理多类别问题。
缺点: 对于输入数据的准备方式较为敏感。
适用数据类型: 标称型数据。
```

朴素贝叶斯 项目案例

项目案例1: 屏蔽社区留言板的侮辱性言论

项目概述

构建一个快速过滤器来屏蔽在线社区留言板上的侮辱性言论。如果某条留言使用了负面或者侮辱性的语言，那么就将该留言标识为内容不当。对此问题建立两个类别: 侮辱类和非侮辱类，使用 1 和 0 分别表示。

开发流程

```
收集数据: 可以使用任何方法
准备数据: 从文本中构建词向量
分析数据: 检查词条确保解析的正确性
训练算法: 从词向量计算概率
测试算法: 根据现实情况修改分类器
使用算法: 对社区留言板言论进行分类
```

> 收集数据: 可以使用任何方法

本例是我们自己构造的词表:

```
def loadDataSet():
    """
    创建数据集
    :return: 单词列表postingList, 所属类别classVec
    """
    postingList = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'], #[0,0,1,1,1......]
                   ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
                   ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                   ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                   ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
                   ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0, 1, 0, 1, 0, 1]  # 1 is abusive, 0 not
    return postingList, classVec
```

> 准备数据: 从文本中构建词向量

```
def createVocabList(dataSet):
    """
    获取所有单词的集合
    :param dataSet: 数据集
    :return: 所有单词的集合(即不含重复元素的单词列表)
    """
    vocabSet = set([])  # create empty set
    for document in dataSet:
```

```
        # 操作符 | 用于求两个集合的并集
        vocabSet = vocabSet | set(document)  # union of the two sets
    return list(vocabSet)


def setOfWords2Vec(vocabList, inputSet):
    """
    遍历查看该单词是否出现，出现该单词则将该单词置1
    :param vocabList: 所有单词集合列表
    :param inputSet: 输入数据集
    :return: 匹配列表[0,1,0,1...]，其中 1与0 表示词汇表中的单词是否出现在输入的数据集中
    """
    # 创建一个和词汇表等长的向量，并将其元素都设置为0
    returnVec = [0] * len(vocabList)# [0,0......]
    # 遍历文档中的所有单词，如果出现了词汇表中的单词，则将输出的文档向量中的对应值设为1
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else:
            print "the word: %s is not in my Vocabulary!" % word
    return returnVec
```

> 分析数据: 检查词条确保解析的正确性

检查函数执行情况，检查词表，不出现重复单词，需要的话，可以对其进行排序。

```
>>> listOPosts, listClasses = bayes.loadDataSet()
>>> myVocabList = bayes.createVocabList(listOPosts)
>>> myVocabList
['cute', 'love', 'help', 'garbage', 'quit', 'I', 'problems', 'is', 'park',
'stop', 'flea', 'dalmation', 'licks', 'food', 'not', 'him', 'buying', 'posting', 'has', 'worthless', 'ate', 'to', 'maybe', 'please',
'stupid', 'so', 'take', 'mr', 'steak', 'my']
```

检查函数有效性。例如：myVocabList 中索引为 2 的元素是什么单词？应该是是 help 。该单词在第一篇文档中出现了，现在检查一下看看它是否出现在第四篇文档中。

```
>>> bayes.setOfWords2Vec(myVocabList, listOPosts[0])
[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1]

>>> bayes.setOfWords2Vec(myVocabList, listOPosts[3])
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

> 训练算法: 从词向量计算概率

现在已经知道了一个词是否出现在一篇文档中，也知道该文档所属的类别。接下来我们重写贝叶斯准则，将之前的 x, y 替换为 w. 粗体的 w 表示这是一个向量，即它由多个值组成。在这个例子中，数值个数与词汇表中的词个数相同。

$$p(c_i \mid \boldsymbol{w}) = \frac{p(\boldsymbol{w} \mid c_i)p(c_i)}{p(\boldsymbol{w})}$$

我们使用上述公式，对每个类计算该值，然后比较这两个概率值的大小。

首先可以通过类别 i (侮辱性留言或者非侮辱性留言)中的文档数除以总的文档数来计算概率 p(ci) 。接下来计算 p(w | ci) ，这里就要用到朴素贝叶斯假设。如果将 w 展开为一个个独立特征，那么就可以将上述概率写作 p(w0, w1, w2...wn | ci) 。这里假设所有词都互相独立，该假设也称作条件独立性假设（例如 A 和 B 两个人抛骰子，概率是互不影响的，也就是相互独立的，A 抛 2点的同时 B 抛 3 点的概率就是 1/6 * 1/6 ），它意味着可以使用 p(w0 | ci)p(w1 | ci)p(w2 | ci)...p(wn | ci) 来计算上述概率，这样就极大地简化了计算的过程。

朴素贝叶斯分类器训练函数

```
def _trainNB0(trainMatrix, trainCategory):
    """
    训练数据原版
    :param trainMatrix: 文件单词矩阵 [[1,0,1,1,1....],[],[]...]
    :param trainCategory: 文件对应的类别[0,1,1,0....]，列表长度等于单词矩阵数，其中的1代表对应的文件是侮辱性文件，0代表不是侮辱性矩阵
    :return:
    """
    # 文件数
    numTrainDocs = len(trainMatrix)
    # 单词数
    numWords = len(trainMatrix[0])
    # 侮辱性文件的出现概率，即trainCategory中所有的1的个数，
    # 代表的就是多少个侮辱性文件，与文件的总数相除就得到了侮辱性文件的出现概率
    pAbusive = sum(trainCategory) / float(numTrainDocs)
    # 构造单词出现次数列表
    p0Num = zeros(numWords) # [0,0,0,.....]
    p1Num = zeros(numWords) # [0,0,0,.....]

    # 整个数据集单词出现总数
    p0Denom = 0.0
```

```
    p1Denom = 0.0
    for i in range(numTrainDocs):
        # 是否是侮辱性文件
        if trainCategory[i] == 1:
            # 如果是侮辱性文件，对侮辱性文件的向量进行加和
            p1Num += trainMatrix[i] #[0,1,1,....] + [0,1,1,....]->[0,2,2,...]
            # 对向量中的所有元素进行求和，也就是计算所有侮辱性文件中出现的单词总数
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    # 类别1，即侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)....]列表
    # 即 在1类别下，每个单词出现的概率
    p1Vect = p1Num / p1Denom# [1,2,3,5]/90->[1/90,...]
    # 类别0，即正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)....]列表
    # 即 在0类别下，每个单词出现的概率
    p0Vect = p0Num / p0Denom
    return p0Vect, p1Vect, pAbusive
```

> 测试算法: 根据现实情况修改分类器

在利用贝叶斯分类器对文档进行分类时，要计算多个概率的乘积以获得文档属于某个类别的概率，即计算 p(w0|1) * p(w1|1) * p(w2|1)。如果其中一个概率值为 0，那么最后的乘积也为 0。为降低这种影响，可以将所有词的出现数初始化为 1，并将分母初始化为 2 （ 取1 或 2 的目的主要是为了保证分子和分母不为 0，大家可以根据业务需求进行更改 ）。

另一个遇到的问题是下溢出，这是由于太多很小的数相乘造成的。当计算乘积 p(w0|ci) * p(w1|ci) * p(w2|ci)... p(wn|ci) 时，由于大部分因子都非常小，所以程序会下溢出或者得到不正确的答案。（ 用 Python 尝试相乘许多很小的数，最后四舍五入后会得到 0 ）。一种解决办法是对乘积取自然对数。在代数中有 ln(a * b) = ln(a) + ln(b), 于是通过求对数可以避免下溢出或者浮点数舍入导致的错误。同时，采用自然对数进行处理不会有任何损失。

下图给出了函数 f(x) 与 ln(f(x)) 的曲线。可以看出，它们在相同区域内同时增加或者减少，并且在相同点上取到极值。它们的取值虽然不同，但不影响最终结果。

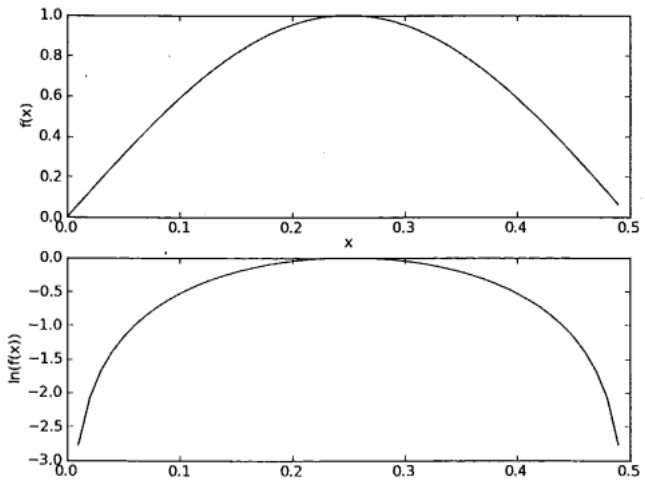


图4-4 函数f(x)与ln(f(x))会一块增大。这表明想求函数的最大值时，可以使用该函数的自然对数来替换原函数进行求解

```
def trainNB0(trainMatrix, trainCategory):
    """
    训练数据优化版本
    :param trainMatrix: 文件单词矩阵
    :param trainCategory: 文件对应的类别
    :return:
    """
    # 总文件数
    numTrainDocs = len(trainMatrix)
    # 总单词数
    numWords = len(trainMatrix[0])
    # 侮辱性文件的出现概率
    pAbusive = sum(trainCategory) / float(numTrainDocs)
    # 构造单词出现次数列表
    # p0Num 正常的统计
    # p1Num 侮辱的统计
    p0Num = ones(numWords)#[0,0......]->[1,1,1,1,1.....]
    p1Num = ones(numWords)

    # 整个数据集单词出现总数，2.0根据样本/实际调查结果调整分母的值（2主要是避免分母为0，当然值可以调整）
    # p0Denom 正常的统计
    # p1Denom 侮辱的统计
    p0Denom = 2.0
    p1Denom = 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
```

```
            # 累加辱骂词的频次
            p1Num += trainMatrix[i]
            # 对每篇文章的辱骂的频次 进行统计汇总
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    # 类别1，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]列表
    p1Vect = log(p1Num / p1Denom)
    # 类别0，即正常文档的[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]列表
    p0Vect = log(p0Num / p0Denom)
    return p0Vect, p1Vect, pAbusive
```

> 使用算法: 对社区留言板言论进行分类

朴素贝叶斯分类函数

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    """
    使用算法：
        # 将乘法转换为加法
        乘法：P(C|F1F2...Fn) = P(F1F2...Fn|C)P(C)/P(F1F2...Fn)
        加法：P(F1|C)*P(F2|C)....P(Fn|C)P(C) -> log(P(F1|C))+log(P(F2|C))+....+log(P(Fn|C))+log(P(C))
    :param vec2Classify: 待测数据[0,1,1,1,1...]，即要分类的向量
    :param p0Vec: 类别0，即正常文档的[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]列表
    :param p1Vec: 类别1，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]列表
    :param pClass1: 类别1，侮辱性文件的出现概率
    :return: 类别1 or 0
    """
    # 计算公式  log(P(F1|C))+log(P(F2|C))+....+log(P(Fn|C))+log(P(C))
    # 大家可能会发现，上面的计算公式，没有除以贝叶斯准则的公式的分母，也就是 P(w) （P(w) 指的是此文档在所有的文档中出现的概率）就进行概率大小的比
    # 因为 P(w) 针对的是包含侮辱和非侮辱的全部文档，所以 P(w) 是相同的。
    # 使用 NumPy 数组来计算两个向量相乘的结果，这里的相乘是指对应元素相乘，即先将两个向量中的第一个元素相乘，然后将第2个元素相乘，以此类推。
    # 我的理解是：这里的 vec2Classify * p1Vec 的意思就是将每个词与其对应的概率相关联起来
    p1 = sum(vec2Classify * p1Vec) + log(pClass1) # P(w|c1) * P(c1) ，即贝叶斯准则的分子
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1) # P(w|c0) * P(c0) ，即贝叶斯准则的分子·
    if p1 > p0:
        return 1
    else:
        return 0


def testingNB():
    """
    测试朴素贝叶斯算法
    """
    # 1. 加载数据集
    listOPosts, listClasses = loadDataSet()
    # 2. 创建单词集合
    myVocabList = createVocabList(listOPosts)
    # 3. 计算单词是否出现并创建数据矩阵
    trainMat = []
    for postinDoc in listOPosts:
        # 返回m*len(myVocabList)的矩阵， 记录的都是0，1信息
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
    # 4. 训练数据
    p0V, p1V, pAb = trainNB0(array(trainMat), array(listClasses))
    # 5. 测试数据
    testEntry = ['love', 'my', 'dalmation']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb)
    testEntry = ['stupid', 'garbage']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb)
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/4.NaiveBayes/bayes.py

项目案例2: 使用朴素贝叶斯过滤垃圾邮件

项目概述

完成朴素贝叶斯的一个最著名的应用: 电子邮件垃圾过滤。

开发流程

使用朴素贝叶斯对电子邮件进行分类

```
收集数据：提供文本文件
准备数据：将文本文件解析成词条向量
分析数据：检查词条确保解析的正确性
```

```
训练算法：使用我们之前建立的 trainNB() 函数
测试算法：使用朴素贝叶斯进行交叉验证
使用算法：构建一个完整的程序对一组文档进行分类，将错分的文档输出到屏幕上
```

> 收集数据: 提供文本文件

文本文件内容如下:

```
Hi Peter,

With Jose out of town, do you want to
meet once in a while to keep things
going and do some interesting stuff?

Let me know
Eugene
```

> 准备数据: 将文本文件解析成词条向量

使用正则表达式来切分文本

```
>>> mySent = 'This book is the best book on Python or M.L. I have ever laid eyes upon.'
>>> import re
>>> regEx = re.compile('\\W*')
>>> listOfTokens = regEx.split(mySent)
>>> listOfTokens
['This', 'book', 'is', 'the', 'best', 'book', 'on', 'Python', 'or', 'M.L.', 'I', 'have', 'ever', 'laid', 'eyes', 'upon', '']
```

> 分析数据: 检查词条确保解析的正确性

> 训练算法: 使用我们之前建立的 trainNB0() 函数

```
def trainNB0(trainMatrix, trainCategory):
    """
    训练数据优化版本
    :param trainMatrix: 文件单词矩阵
    :param trainCategory: 文件对应的类别
    :return:
    """
    # 总文件数
    numTrainDocs = len(trainMatrix)
    # 总单词数
    numWords = len(trainMatrix[0])
    # 侮辱性文件的出现概率
    pAbusive = sum(trainCategory) / float(numTrainDocs)
    # 构造单词出现次数列表
    # p0Num 正常的统计
    # p1Num 侮辱的统计
    p0Num = ones(numWords)#[0,0......]->[1,1,1,1,1.....]
    p1Num = ones(numWords)

    # 整个数据集单词出现总数，2.0根据样本/实际调查结果调整分母的值（2主要是避免分母为0，当然值可以调整）
    # p0Denom 正常的统计
    # p1Denom 侮辱的统计
    p0Denom = 2.0
    p1Denom = 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            # 累加辱骂词的频次
            p1Num += trainMatrix[i]
            # 对每篇文章的辱骂的频次 进行统计汇总
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    # 类别1，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]列表
    p1Vect = log(p1Num / p1Denom)
    # 类别0，即正常文档的[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]列表
    p0Vect = log(p0Num / p0Denom)
    return p0Vect, p1Vect, pAbusive
```

> 测试算法: 使用朴素贝叶斯进行交叉验证

文件解析及完整的垃圾邮件测试函数

```
# 切分文本
def textParse(bigString):
    '''
    Desc:
        接收一个大字符串并将其解析为字符串列表
```

```
    Args:
        bigString -- 大字符串
    Returns:
        去掉少于 2 个字符的字符串，并将所有字符串转换为小写，返回字符串列表
    '''
    import re
    # 使用正则表达式来切分句子，其中分隔符是除单词、数字外的任意字符串
    listOfTokens = re.split(r'\W*', bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]

def spamTest():
    '''
    Desc:
        对贝叶斯垃圾邮件分类器进行自动化处理。
    Args:
        none
    Returns:
        对测试集中的每封邮件进行分类，若邮件分类错误，则错误数加 1，最后返回总的错误百分比。
    '''
    docList = []
    classList = []
    fullText = []
    for i in range(1, 26):
        # 切分，解析数据，并归类为 1 类别
        wordList = textParse(open('input/4.NaiveBayes/email/spam/%d.txt' % i).read())
        docList.append(wordList)
        classList.append(1)
        # 切分，解析数据，并归类为 0 类别
        wordList = textParse(open('input/4.NaiveBayes/email/ham/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    # 创建词汇表
    vocabList = createVocabList(docList)
    trainingSet = range(50)
    testSet = []
    # 随机取 10 个邮件用来测试
    for i in range(10):
        # random.uniform(x, y) 随机生成一个范围为 x - y 的实数
        randIndex = int(random.uniform(0, len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat = []
    trainClasses = []
    for docIndex in trainingSet:
        trainMat.append(setOfWords2Vec(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V, p1V, pSpam = trainNB0(array(trainMat), array(trainClasses))
    errorCount = 0
    for docIndex in testSet:
        wordVector = setOfWords2Vec(vocabList, docList[docIndex])
        if classifyNB(array(wordVector), p0V, p1V, pSpam) != classList[docIndex]:
            errorCount += 1
    print 'the errorCount is: ', errorCount
    print 'the testSet length is :', len(testSet)
    print 'the error rate is :', float(errorCount)/len(testSet)
```

> 使用算法: 构建一个完整的程序对一组文档进行分类，将错分的文档输出到屏幕上

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/4.NaiveBayes/bayes.py

项目案例3: 使用朴素贝叶斯分类器从个人广告中获取区域倾向

项目概述

广告商往往想知道关于一个人的一些特定人口统计信息，以便能更好地定向推销广告。

我们将分别从美国的两个城市中选取一些人，通过分析这些人发布的信息，来比较这两个城市的人们在广告用词上是否不同。如果结论确实不同，那么他们各自常用的词是那些，从人们的用词当中，我们能否对不同城市的人所关心的内容有所了解。

开发流程

```
收集数据: 从 RSS 源收集内容，这里需要对 RSS 源构建一个接口
准备数据: 将文本文件解析成词条向量
分析数据: 检查词条确保解析的正确性
训练算法: 使用我们之前简历的 trainNB0() 函数
测试算法: 观察错误率，确保分类器可用。可以修改切分程序，以降低错误率，提高分类结果
使用算法: 构建一个完整的程序，封装所有内容。给定两个 RSS 源，改程序会显示最常用的公共词
```

> 收集数据: 从 RSS 源收集内容，这里需要对 RSS 源构建一个接口

也就是导入 RSS 源，我们使用 python 下载文本，在http://code.google.com/p/feedparser/ 下浏览相关文档，安装 feedparse，首先解压下载的包，并将当前目录切换到解压文件所在的文件夹，然后在 python 提示符下输入：

```
>>> python setup.py install
```

> 准备数据: 将文本文件解析成词条向量

文档词袋模型

我们将每个词的出现与否作为一个特征，这可以被描述为 词集模型(set-of-words model) 。如果一个词在文档中出现不止一次，这可能意味着包含该词是否出现在文档中所不能表达的某种信息，这种方法被称为 词袋模型(bag-of-words model) 。在词袋中，每个单词可以出现多次，而在词集中，每个词只能出现一次。为适应词袋模型，需要对函数 setOfWords2Vec() 稍加修改，修改后的函数为 bagOfWords2Vec() 。

如下给出了基于词袋模型的朴素贝叶斯代码。它与函数 setOfWords2Vec() 几乎完全相同，唯一不同的是每当遇到一个单词时，它会增加词向量中的对应值，而不只是将对应的数值设为 1 。

```
def bagOfWords2VecMN(vocaList, inputSet):
    returnVec = [0] * len(vocabList)
    for word in inputSet:
        if word in inputSet:
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

```
#创建一个包含在所有文档中出现的不重复词的列表
def createVocabList(dataSet):
    vocabSet=set([])    #创建一个空集
    for document in dataSet:
        vocabSet=vocabSet|set(document)   #创建两个集合的并集
    return list(vocabSet)
def setOfWords2VecMN(vocabList,inputSet):
    returnVec=[0]*len(vocabList)  #创建一个其中所含元素都为0的向量
    for word in inputSet:
        if word in vocabList:
                returnVec[vocabList.index(word)]+=1
    return returnVec

#文件解析
def textParse(bigString):
    import re
    listOfTokens=re.split(r'\W*',bigString)
    return [tok.lower() for tok in listOfTokens if len(tok)>2]
```

> 分析数据: 检查词条确保解析的正确性

> 训练算法: 使用我们之前简历的 trainNB0() 函数

```
def trainNB0(trainMatrix, trainCategory):
    """
    训练数据优化版本
    :param trainMatrix: 文件单词矩阵
    :param trainCategory: 文件对应的类别
    :return:
    """
    # 总文件数
    numTrainDocs = len(trainMatrix)
    # 总单词数
    numWords = len(trainMatrix[0])
    # 侮辱性文件的出现概率
    pAbusive = sum(trainCategory) / float(numTrainDocs)
    # 构造单词出现次数列表
    # p0Num 正常的统计
    # p1Num 侮辱的统计
    # 避免单词列表中的任何一个单词为0，而导致最后的乘积为0，所以将每个单词的出现次数初始化为 1
    p0Num = ones(numWords)#[0,0......]->[1,1,1,1,1.....]
    p1Num = ones(numWords)

    # 整个数据集单词出现总数，2.0根据样本/实际调查结果调整分母的值（2主要是避免分母为0，当然值可以调整）
    # p0Denom 正常的统计
    # p1Denom 侮辱的统计
    p0Denom = 2.0
    p1Denom = 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            # 累加辱骂词的频次
            p1Num += trainMatrix[i]
            # 对每篇文章的辱骂的频次 进行统计汇总
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
```

```
    # 类别1，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]列表
    p1Vect = log(p1Num / p1Denom)
    # 类别0，即正常文档的[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]列表
    p0Vect = log(p0Num / p0Denom)
    return p0Vect, p1Vect, pAbusive
```

> 测试算法: 观察错误率，确保分类器可用。可以修改切分程序，以降低错误率，提高分类结果

```
#RSS源分类器及高频词去除函数
def calcMostFreq(vocabList,fullText):
    import operator
    freqDict={}
    for token in vocabList:  #遍历词汇表中的每个词
        freqDict[token]=fullText.count(token)  #统计每个词在文本中出现的次数
    sortedFreq=sorted(freqDict.iteritems(),key=operator.itemgetter(1),reverse=True)  #根据每个词出现的次数从高到底对字典进行排序
    return sortedFreq[:30]   #返回出现次数最高的30个单词
def localWords(feed1,feed0):
    import feedparser
    docList=[];classList=[];fullText=[]
    minLen=min(len(feed1['entries']),len(feed0['entries']))
    for i in range(minLen):
        wordList=textParse(feed1['entries'][i]['summary'])   #每次访问一条RSS源
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1)
        wordList=textParse(feed0['entries'][i]['summary'])
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList=createVocabList(docList)
    top30Words=calcMostFreq(vocabList,fullText)
    for pairW in top30Words:
        if pairW[0] in vocabList:vocabList.remove(pairW[0])    #去掉出现次数最高的那些词
    trainingSet=range(2*minLen);testSet=[]
    for i in range(20):
        randIndex=int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[];trainClasses=[]
    for docIndex in trainingSet:
        trainMat.append(bagOfWords2VecMN(vocabList,docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam=trainNBO(array(trainMat),array(trainClasses))
    errorCount=0
    for docIndex in testSet:
        wordVector=bagOfWords2VecMN(vocabList,docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam)!=classList[docIndex]:
            errorCount+=1
    print 'the error rate is:',float(errorCount)/len(testSet)
    return vocabList,p0V,p1V

#朴素贝叶斯分类函数
def classifyNB(vec2Classify,p0Vec,p1Vec,pClass1):
    p1=sum(vec2Classify*p1Vec)+log(pClass1)
    p0=sum(vec2Classify*p0Vec)+log(1.0-pClass1)
    if p1>p0:
        return 1
    else:
        return 0
```

> 使用算法: 构建一个完整的程序，封装所有内容。给定两个 RSS 源，改程序会显示最常用的公共词

函数 localWords() 使用了两个 RSS 源作为参数，RSS 源要在函数外导入，这样做的原因是 RSS 源会随时间而改变，重新加载 RSS 源就会得到新的数据

```
>>> reload(bayes)
<module 'bayes' from 'bayes.pyc'>
>>> import feedparser
>>> ny=feedparser.parse('http://newyork.craigslist.org/stp/index.rss')
>>> sy=feedparser.parse('http://sfbay.craigslist.org/stp/index.rss')
>>> vocabList,pSF,pNY=bayes.localWords(ny,sf)
the error rate is: 0.2
>>> vocabList,pSF,pNY=bayes.localWords(ny,sf)
the error rate is: 0.3
>>> vocabList,pSF,pNY=bayes.localWords(ny,sf)
the error rate is: 0.55
```

为了得到错误率的精确估计，应该多次进行上述实验，然后取平均值

接下来，我们要分析一下数据，显示地域相关的用词

可以先对向量pSF与pNY进行排序，然后按照顺序打印出来，将下面的代码添加到文件中：

```
#最具表征性的词汇显示函数
def getTopWords(ny,sf):
    import operator
    vocabList,p0V,p1V=localWords(ny,sf)
    topNY=[];topSF=[]
    for i in range(len(p0V)):
        if p0V[i]>-6.0:topSF.append((vocabList[i],p0V[i]))
        if p1V[i]>-6.0:topNY.append((vocabList[i],p1V[i]))
    sortedSF=sorted(topSF,key=lambda pair:pair[1],reverse=True)
    print "SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**"
    for item in sortedSF:
        print item[0]
    sortedNY=sorted(topNY,key=lambda pair:pair[1],reverse=True)
    print "NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**"
    for item in sortedNY:
        print item[0]
```

函数 getTopWords() 使用两个 RSS 源作为输入，然后训练并测试朴素贝叶斯分类器，返回使用的概率值。然后创建两个列表用于元组的存储，与之前返回排名最高的 X 个单词不同，这里可以返回大于某个阈值的所有词，这些元组会按照它们的条件概率进行排序。

保存 bayes.py 文件，在python提示符下输入：

```
>>> reload(bayes)
<module 'bayes' from 'bayes.pyc'>
>>> bayes.getTopWords(ny,sf)
the error rate is: 0.55
SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**
how
last
man
...
veteran
still
ends
late
off
own
know
NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**
someone
meet
...
apparel
recalled
starting
strings
```

当注释掉用于移除高频词的三行代码，然后比较注释前后的分类性能，去掉这几行代码之后，错误率为54%，，而保留这些代码得到的错误率为70%。这里观察到，这些留言中出现次数最多的前30个词涵盖了所有用词的30%，vocabList的大小约为3000个词，也就是说，词汇表中的一小部分单词却占据了所有文本用词的一大部分。产生这种现象的原因是因为语言中大部分都是冗余和结构辅助性内容。另一个常用的方法是不仅移除高频词，同时从某个预定高频词中移除结构上的辅助词，该词表称为停用词表。

最后输出的单词，可以看出程序输出了大量的停用词，可以移除固定的停用词看看结果如何，这样做的花，分类错误率也会降低。

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/4.NaiveBayes/bayes.py

---

第5章 Logistic 回归

Logistic 回归 概述

`Logistic` 回归虽然名字叫回归，但是它是用来做分类的。其主要思想是：根据现有数据对分类边界线建立回归公式，以此进行分类。

须知概念

Sigmoid 函数

回归 概念

假设现在有一些数据点，我们用一条直线对这些点进行拟合（这条直线称为最佳拟合直线），这个拟合的过程就叫做回归。进而可以得到对这些点的拟合直线方程，那么我们根据这个回归方程，怎么进行分类呢？请看下面。

二值型输出分类函数

我们想要的函数应该是: 能接受所有的输入然后预测出类别。例如，在两个类的情况下，上述函数输出 0 或 1.或许你之前接触过具有这种性质的函数，该函数称为 `海维塞得阶跃函数(Heaviside step function)`，或者直接称为 `单位阶跃函数`。然而，海维塞得阶跃函数的问题在于: 该函数在跳跃点上从 0 瞬间跳跃到 1，这个瞬间跳跃过程有时很难处理。幸好，另一个函数也有类似的性质（可以输出 0 或者 1 的性质），且数学上更易处理，这就是 Sigmoid 函数。
Sigmoid 函数具体的计算公式如下:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

下图给出了 Sigmoid 函数在不同坐标尺度下的两条曲线图。当 x 为 0 时，Sigmoid 函数值为 0.5 。随着 x 的增大，对应的 Sigmoid 值将逼近于 1 ; 而随着 x 的减小， Sigmoid 值将逼近于 0 。如果横坐标刻度足够大， Sigmoid 函数看起来很像一个阶跃函数。

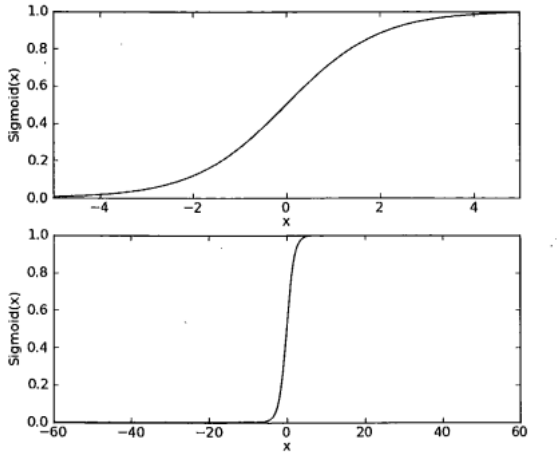


图5-1　两种坐标尺度下的Sigmoid函数图。上图的横坐标为–5到5，这时的曲线变化较为平滑；下图横坐标的尺度足够大，可以看到，在x＝0点处Sigmoid函数看起来很像阶跃函数

因此，为了实现 Logistic 回归分类器，我们可以在每个特征上都乘以一个回归系数（如下公式所示），然后把所有结果值相加，将这个总和代入 Sigmoid 函数中，进而得到一个范围在 0~1 之间的数值。任何大于 0.5 的数据被分入 1 类，小于 0.5 即被归入 0 类。所以， Logistic 回归也可以被看成是一种概率估计。

基于最优化方法的回归系数确定

Sigmoid 函数的输入记为 z ，由下面公式得到:

$$z = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$$

如果采用向量的写法，上述公式可以写成 $z = w^Tx$ ，它表示将这两个数值向量对应元素相乘然后全部加起来即得到 z 值。其中的向量 x 是分类器的输入数据，向量 w 也就是我们要找到的最佳参数（系数），从而使得分类器尽可能地精确。为了寻找该最佳参数，需要用到最优化理论的一些知识。我们这里使用的是——梯度上升法。

梯度上升法

梯度的介绍

需要一点点向量方面的数学基础

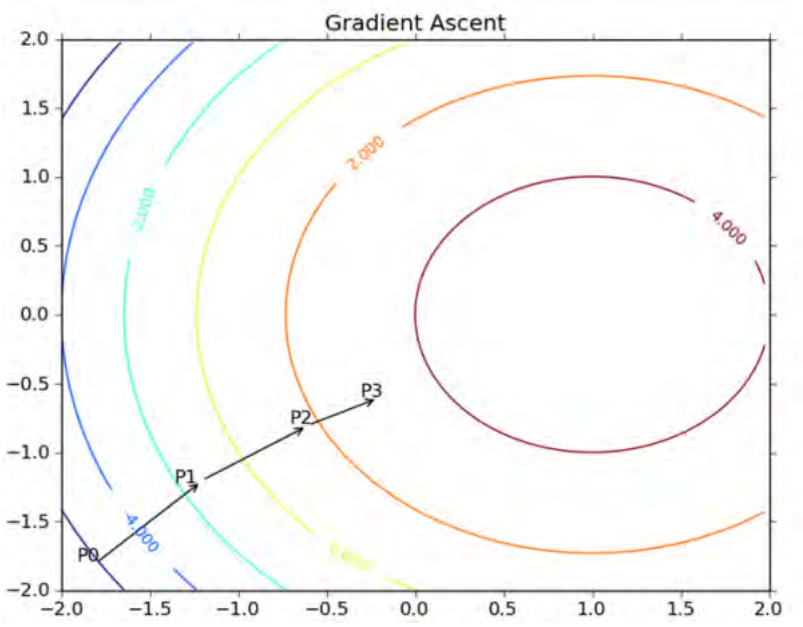
```
向量 = 值 + 方向
梯度 = 向量
梯度 = 梯度值 + 梯度方向
```

梯度上升法的思想

要找到某函数的最大值，最好的方法是沿着该函数的梯度方向探寻。如果梯度记为 ▽ ，则函数 f(x, y) 的梯度由下式表示:

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

这个梯度意味着要沿 x 的方向移动 $\frac{\partial f(x,y))}{\partial x}$ ，沿 y 的方向移动 $\frac{\partial f(x,y))}{\partial y}$ 。其中，函数f(x, y) 必须要在待计算的点上有定义并且可微。下图是一个具体的例子。



上图展示的，梯度上升算法到达每个点后都会重新估计移动的方向。从 P0 开始，计算完该点的梯度，函数就根据梯度移动到下一点 P1。在 P1 点，梯度再次被重新计算，并沿着新的梯度方向移动到 P2 。如此循环迭代，直到满足停止条件。迭代过程中，梯度算子总是保证我们能选取到最佳的移动方向。

上图中的梯度上升算法沿梯度方向移动了一步。可以看到，梯度算子总是指向函数值增长最快的方向。这里所说的是移动方向，而未提到移动量的大小。该量值称为步长，记作 α 。用向量来表示的话，梯度上升算法的迭代公式如下:

$$\boldsymbol{w} := \boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} f(w)$$

该公式将一直被迭代执行，直至达到某个停止条件为止，比如迭代次数达到某个指定值或者算法达到某个可以允许的误差范围。

介绍一下几个相关的概念：

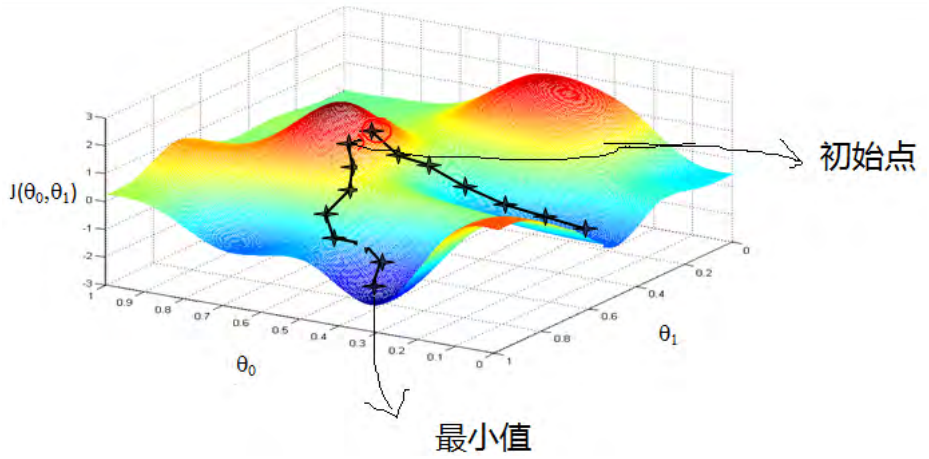
```
例如：y = w1x1 + w2x2 + ... + wnxn
梯度：参考上图的例子，二维图像，x方向代表第一个系数，也就是 w1，y方向代表第二个系数也就是 w2，这样的向量就是梯度。
α：上面的梯度算法的迭代公式中的阿尔法，这个代表的是移动步长。移动步长会影响最终结果的拟合程度，最好的方法就是随着迭代次数更改移动步长。
步长通俗的理解，100米，如果我一步走10米，我需要走10步；如果一步走20米，我只需要走5步。这里的一步走多少米就是步长的意思。
▽f(w)：代表沿着梯度变化的方向。
```

Note: 我们常听到的是梯度下降算法，它与这里的梯度上升算法是一样的，只是公式中的加法需要变成减法。因此，对应的公式可以写成

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} f(w)$$

梯度上升算法用来求函数的最大值，而梯度下降算法用来求函数的最小值。

局部最优现象



上图表示参数 θ 与误差函数 J(θ) 的关系图，红色的部分是表示 J(θ) 有着比较高的取值，我们需要的是，能够让 J(θ) 的值尽量的低。也就是深蓝色的部分。θ0，θ1 表示 θ 向量的两个维度。

可能梯度下降的最终点并非是全局最小点，可能是一个局部最小点，如我们上图中的右边的梯度下降曲线，描述的是最终到达一个局部最小点，这是我们重新选择了一个初始点得到的。

看来我们这个算法将会在很大的程度上被初始点的选择影响而陷入局部最小点。

Logistic 回归 原理

Logistic 回归 工作原理

```
每个回归系数初始化为 1
重复 R 次:
    计算整个数据集的梯度
    使用 步长 x 梯度 更新回归系数的向量
返回回归系数
```

Logistic 回归 开发流程

```
收集数据：采用任意方法收集数据
准备数据：由于需要进行距离计算，因此要求数据类型为数值型。另外，结构化数据格式则最佳。
分析数据：采用任意方法对数据进行分析。
训练算法：大部分时间将用于训练，训练的目的是为了找到最佳的分类回归系数。
测试算法：一旦训练步骤完成，分类将会很快。
使用算法：首先，我们需要输入一些数据，并将其转换成对应的结构化数值；接着，基于训练好的回归系数就可以对这些数值进行简单的回归计算，判定它们属于哪个类别；在这之
```

Logistic 回归 算法特点

```
优点：计算代价不高，易于理解和实现。
缺点：容易欠拟合，分类精度可能不高。
适用数据类型：数值型和标称型数据。
```

附加 方向导数与梯度

第七节 方向导数与梯度

教学目的：掌握方向导数的定义和求法；掌握梯度的定义，求法及其与等高线的关系
教学重点：方向导数与梯度的求法
教学难点：方向角的确定
教学内容：

一、方向导数

现在我们来讨论函数 $z=f(x,y)$ 在一点 P 沿某一方向的变化率问题

定义 设函数 $z=f(x,y)$ 在点 $P(x,y)$ 的某一邻域 $U(P)$ 内有定义，自点 P 引射线 l，设 x 轴正向到射线 l 的转角为 φ（逆时针方向：$\varphi>0$；顺时针方向：$\varphi<0$），并设 $P'(x+\Delta x, y+\Delta y)$ 为 l 上的另一点且 $P'\in U(P)$，我们考虑函数的增量 $f(x+\Delta x, y+\Delta y)-f(x,y)$ 与 P、P' 两点间的距离 $\rho=\sqrt{(\Delta x)^2+(\Delta y)^2}$ 的比值（当 P' 沿着 l 趋于 P 时），如果这个比的极限存在，则称这极限为函数 $f(x,y)$ 在点 P 沿方向 l 的方向导数，记作 $\frac{\partial f}{\partial l}$，即

$$\frac{\partial f}{\partial l}=\lim_{\rho\to 0}\frac{f(x+\Delta x,y+\Delta y)-f(x,y)}{\rho} \tag{1}$$

从定义可知，函数 $f(x,y)$ 在点 $P(x,y)$ 的偏导数 f_x、f_y 存在时，函数在点 P 沿着 x 轴正向 $e_1=\{1,0\}$、y 轴正向 $e_2=\{0,1\}$ 的方向导数存在且其值依次为 f_x、f_y；函数 $f(x,y)$ 在点 P 沿 x 轴负向 $e_1'=\{-1,0\}$、y 轴负向 $e_2'=\{0,-1\}$ 的方向导数也存在且其值依次为 $-f_x$、$-f_y$。

关于方向导数 $\frac{\partial f}{\partial l}$ 的存在及计算，我们有下面的定理

定理 如果函数 $z=f(x,y)$ 在点 $P(x,y)$ 是可微分的，那末函数在该点沿任一方向的方向导数都存在，且有

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi, \tag{2}$$

其中 φ 为 x 轴到方向 l 的转角

证 根据函数 $z=f(x,y)$ 在点 $P(x,y)$ 可微分的假定，函数的增量可以表达为

$$f(x+\Delta x,y+\Delta y)-f(x,y)=\frac{\partial f}{\partial x}\Delta x+\frac{\partial f}{\partial y}\Delta y+o(\rho).$$

两边各除以 ρ，得到

$$\begin{aligned}&\frac{f(x+\Delta x,y+\Delta y)-f(x,y)}{\rho}\\&=\frac{\partial f}{\partial x}\cdot\frac{\Delta x}{\rho}+\frac{\partial f}{\partial y}\cdot\frac{\Delta y}{\rho}+\frac{o(\rho)}{\rho}\\&=\frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi+\frac{o(\rho)}{\rho}\end{aligned}$$

所以

$$\lim_{\rho\to 0}\frac{f(x+\Delta x,y+\Delta y)-f(x,y)}{\rho}=\frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi$$

这就证明了方向导数存在且其值为

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi.$$

例6 求函数 $z=xe^{2y}$ 在点 $P(1,0)$ 处沿从点 $P(1,0)$ 到点 $Q(2,-1)$ 方向的方向导数

解 这里方向 l 即向量 $\overrightarrow{PQ}=\{1,-1\}$ 的方向，因此 x 轴到方向 l 的转角 $\varphi=-\frac{\pi}{4}$，

因为 $\frac{\partial z}{\partial x}=e^{2y}$，$\frac{\partial z}{\partial y}=2xe^{2y}$，

在点 $(1,0)$，$\frac{\partial z}{\partial x}=1$，$\frac{\partial z}{\partial y}=2$，

故所求方向导数

$$\frac{\partial z}{\partial l}=1\cdot\cos\left(-\frac{\pi}{4}\right)+2\sin\left(-\frac{\pi}{4}\right)=-\frac{\sqrt{2}}{2}$$

例7 设由原点到点 (x,y) 的向径为 r，x 轴到 r 的转角为 θ，x 轴到射线 l 的转角为 φ，求 $\frac{\partial r}{\partial l}$，其中 $r=|r|=\sqrt{x^2+y^2}$ $(r\neq 0)$

解 因为

$$\frac{\partial r}{\partial x}=\frac{x}{\sqrt{x^2+y^2}}=\frac{x}{r}=\cos\theta,$$

$$\frac{\partial r}{\partial y}=\frac{y}{\sqrt{x^2+y^2}}=\frac{y}{r}=\sin\theta$$

所以

$$\frac{\partial r}{\partial l}=\cos\theta\cos\varphi+\sin\theta\sin\varphi=\cos(\theta-\varphi)$$

由例6.26可知，当 $\varphi=\theta$ 时，$\frac{\partial r}{\partial l}=1$，即 r 沿着与向径本身方向的方向导数为1；而当 $\varphi=\theta\pm\frac{\pi}{2}$ 时，$\frac{\partial r}{\partial l}=0$，即 r 沿着与向径垂直方向的方向导数为零

对于三元函数 $u=f(x,y,z)$ 来说，它在空间一点 $P(x,y,z)$ 沿着方向 l（设方向 l 的方向角为 α、β、γ）的方向导数，同样可以定义为

$$\frac{\partial f}{\partial l}=\lim_{\rho\to 0}\frac{f(x+\Delta x,y+\Delta y,z+\Delta z)-f(x,y,z)}{\rho}, \tag{3}$$

其中 $\rho=\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2}$，$\Delta x=\rho\cos\alpha$，$\Delta y=\rho\cos\beta$，$\Delta z=\rho\cos\gamma$

同样可以证明，如果函数在所考虑的点处可微分，那末函数在该点沿着方向 l 的方向导数为

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\cos\beta+\frac{\partial f}{\partial z}\cos\gamma.$$

二、梯度

1.梯度的定义

与方向导数有关联的一个概念是函数的梯度

定义 设函数 $z=f(x,y)$ 在平面区域 D 内具有一阶连续偏导数，则对于每一点 $(x,y)\in D$，都可定出一个向量

$$\frac{\partial f}{\partial x}i+\frac{\partial f}{\partial y}j,$$

这向量称为函数 $z=f(x,y)$ 在点 $P(x,y)$ 的梯度，记作 $\mathrm{grad}\,f(x,y)$，即

$$\mathrm{grad}\,f(x,y)=\frac{\partial f}{\partial x}i+\frac{\partial f}{\partial y}j.$$

如果设 $e=\cos\varphi i+\sin\varphi j$ 是与方向 l 同方向的单位向量，则由方向导数的计算公式可知

$$\begin{aligned}\frac{\partial f}{\partial l}&=\frac{\partial f}{\partial x}\cos\varphi+\frac{\partial f}{\partial y}\sin\varphi=\left\{\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\right\}\cdot\{\cos\varphi,\sin\varphi\}\\&=\mathrm{grad}f(x,y)\cdot e\\&=|\mathrm{grad}f(x,y)|\cos(\mathrm{grad}f(x,y)^\wedge,e)\end{aligned}$$

这里，$(\mathrm{grad}\,f(x,y)^\wedge,e)$ 表示向量 $\mathrm{grad}\,f(x,y)$ 与 e 的夹角，由此可以看出，就是梯度在射线 l 上的投影，当方向 l 与梯度的方向一致时，有

$$\cos(\mathrm{grad}\,f(x,y)^\wedge,e)=1,$$

Logistic 回归 项目案例

项目案例1: 使用 Logistic 回归在简单数据集上的分类

项目概述

在一个简单的数据集上，采用梯度上升法找到 Logistic 回归分类器在此数据集上的最佳回归系数

开发流程

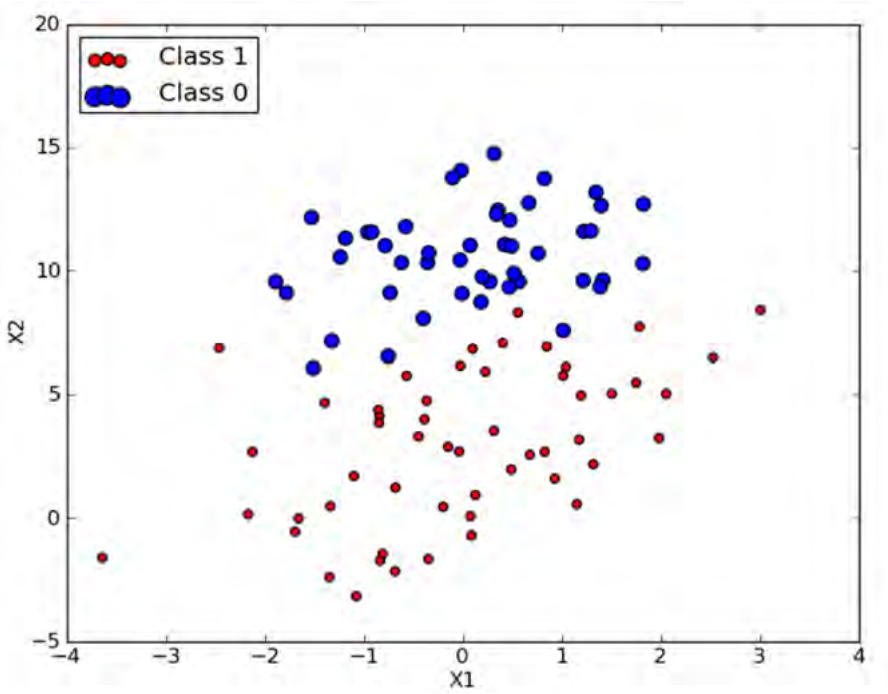
```
收集数据：可以使用任何方法
准备数据：由于需要进行距离计算，因此要求数据类型为数值型。另外，结构化数据格式则最佳
分析数据：画出决策边界
训练算法：使用梯度上升找到最佳参数
测试算法：使用 Logistic 回归进行分类
使用算法：对简单数据集中数据进行分类
```

> 收集数据: 可以使用任何方法

我们采用存储在 TestSet.txt 文本文件中的数据，存储格式如下:

```
-0.017612       14.053064       0
-1.395634       4.662541        1
-0.752157       6.538620        0
-1.322371       7.152853        0
0.423363        11.054677       0
```

绘制在图中，如下图所示:



> 准备数据: 由于需要进行距离计算，因此要求数据类型为数值型。另外，结构化数据格式则最佳

> 分析数据: 画出决策边界

画出数据集和 Logistic 回归最佳拟合直线的函数

```
def plotBestFit(dataArr, labelMat, weights):
    '''
        Desc:
            将我们得到的数据可视化展示出来
        Args:
            dataArr:样本数据的特征
            labelMat:样本数据的类别标签，即目标变量
            weights:回归系数
        Returns:
            None
    '''

    n = shape(dataArr)[0]
    xcord1 = []; ycord1 = []
    xcord2 = []; ycord2 = []
    for i in range(n):
        if int(labelMat[i])== 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')
    ax.scatter(xcord2, ycord2, s=30, c='green')
    x = arange(-3.0, 3.0, 0.1)
    """
    y的由来，卧槽，是不是没看懂？
    首先理论上是这个样子的。
    dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
    w0*x0+w1*x1+w2*x2=f(x)
    x0最开始就设置为1叻， x2就是我们画图的y值，而f(x)被我们磨合误差给算到w0,w1,w2身上去了
    所以： w0+w1*x+w2*y=0 => y = (-w0-w1*x)/w2
    """
    y = (-weights[0]-weights[1]*x)/weights[2]
    ax.plot(x, y)
    plt.xlabel('X'); plt.ylabel('Y')
    plt.show()
```

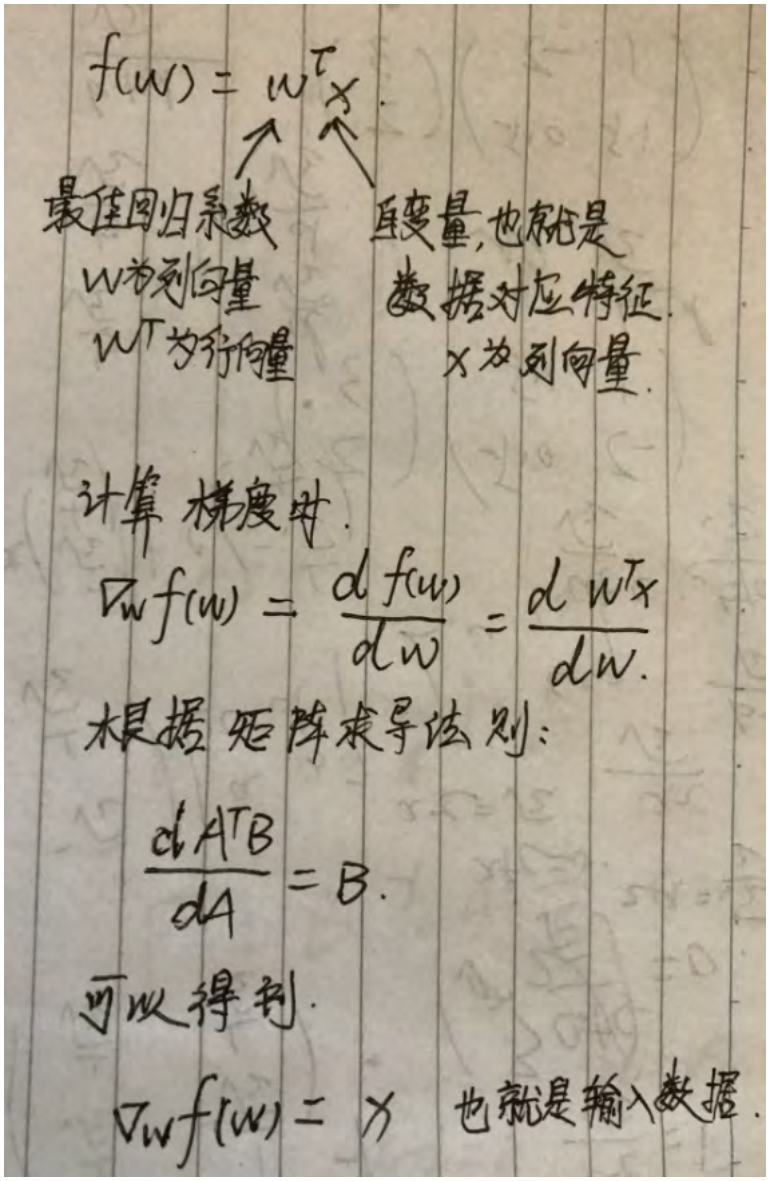
> 训练算法: 使用梯度上升找到最佳参数

Logistic 回归梯度上升优化算法

```
# 正常的处理方案
# 两个参数：第一个参数==> dataMatIn 是一个2维NumPy数组，每列分别代表每个不同的特征，每行则代表每个训练样本。
# 第二个参数==> classLabels 是类别标签，它是一个 1*100 的行向量。为了便于矩阵计算，需要将该行向量转换为列向量，做法是将原向量转置，再将它赋值给
def gradAscent(dataMatIn, classLabels):
    # 转化为矩阵[[1,1,2],[1,1,2]....]
```

```
    dataMatrix = mat(dataMatIn)             # 转换为 NumPy 矩阵
    # 转化为矩阵[[0,1,0,1,0,1.....]]，并转制[[0],[1],[0].....]
    # transpose() 行列转置函数
    # 将行向量转化为列向量    =>  矩阵的转置
    labelMat = mat(classLabels).transpose() # 首先将数组转换为 NumPy 矩阵，然后再将行向量转置为列向量
    # m->数据量，样本数 n->特征数
    m,n = shape(dataMatrix)
    # print m, n, '__'*10, shape(dataMatrix.transpose()), '__'*100
    # alpha代表向目标移动的步长
    alpha = 0.001
    # 迭代次数
    maxCycles = 500
    # 生成一个长度和特征数相同的矩阵，此处n为3 -> [[1],[1],[1]]
    # weights 代表回归系数， 此处的 ones((n,1)) 创建一个长度和特征数相同的矩阵，其中的数全部都是 1
    weights = ones((n,1))
    for k in range(maxCycles):              #heavy on matrix operations
        # m*3 的矩阵 * 3*1 的单位矩阵 = m*1的矩阵
        # 那么乘上单位矩阵的意义，就代表：通过公式得到的理论值
        # 参考地址： 矩阵乘法的本质是什么？ https://www.zhihu.com/question/21351965/answer/31050145
        # print 'dataMatrix====', dataMatrix
        # print 'weights====', weights
        # n*3   *  3*1  = n*1
        h = sigmoid(dataMatrix*weights)     # 矩阵乘法
        # print 'hhhhhhh====', h
        # labelMat是实际值
        error = (labelMat - h)              # 向量相减
        # 0.001* (3*m)*(m*1) 表示在每一个列上的一个误差情况，最后得出 x1,x2,xn的系数的偏移量
        weights = weights + alpha * dataMatrix.transpose() * error # 矩阵乘法，最后得到回归系数
    return array(weights)
```

大家看到这儿可能会有一些疑惑，就是，我们在迭代中更新我们的回归系数，后边的部分是怎么计算出来的？为什么会是 alpha * dataMatrix.transpose() * error ?因为这就是我们所求的梯度，也就是对 f(w) 对 w 求一阶导数。具体推导如下:



> 测试算法: 使用 Logistic 回归进行分类

```
def testLR():
    # 1.收集并准备数据
    dataMat, labelMat = loadDataSet("input/5.Logistic/TestSet.txt")

    # print dataMat, '---\n', labelMat
    # 2.训练模型，  f(x)=a1*x1+b2*x2+..+nn*xn中 (a1,b2, .., nn).T的矩阵值
    # 因为数组没有是复制n份， array的乘法就是乘法
    dataArr = array(dataMat)
    # print dataArr
    weights = gradAscent(dataArr, labelMat)
    # weights = stocGradAscent0(dataArr, labelMat)
    # weights = stocGradAscent1(dataArr, labelMat)
    # print '*'*30, weights

    # 数据可视化
    plotBestFit(dataArr, labelMat, weights)
```

> 使用算法: 对简单数据集中数据进行分类

注意

梯度上升算法在每次更新回归系数时都需要遍历整个数据集，该方法在处理 100 个左右的数据集时尚可，但如果有数十亿样本和成千上万的特征，那么该方法的计算复杂度就太高了。一种改进方法是一次仅用一个样本点来更新回归系数，该方法称为 随机梯度上升算法。由于可以在新样本到来时对分类器进行增量式更新，因而随机梯度上升算法是一个在线学习算法。与 “在线学习” 相对应，一次处理所有数据被称作是 “批处理”。

随机梯度上升算法可以写成如下的伪代码:

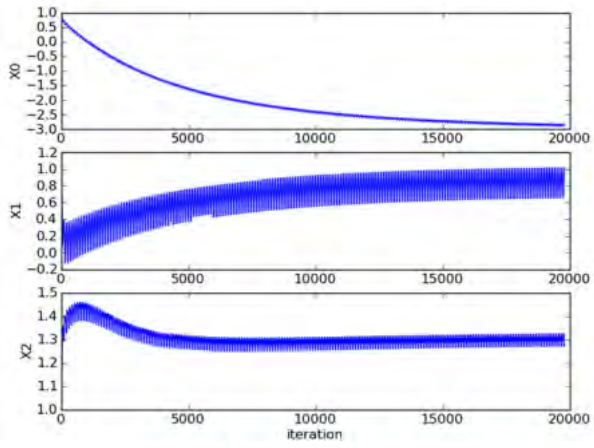
```
所有回归系数初始化为 1
对数据集中每个样本
    计算该样本的梯度
    使用 alpha x gradient 更新回归系数值
返回回归系数值
```

以下是随机梯度上升算法的实现代码:

```
# 随机梯度上升
# 梯度上升优化算法在每次更新数据集时都需要遍历整个数据集，计算复杂都较高
# 随机梯度上升一次只用一个样本点来更新回归系数
def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.01
    # n*1的矩阵
    # 函数ones创建一个全1的数组
    weights = ones(n)   # 初始化长度为n的数组，元素全部为 1
    for i in range(m):
        # sum(dataMatrix[i]*weights)为了求 f(x)的值， f(x)=a1*x1+b2*x2+..+nn*xn,此处求出的 h 是一个具体的数值，而不是一个矩阵
        h = sigmoid(sum(dataMatrix[i]*weights))
        # print 'dataMatrix[i]===', dataMatrix[i]
        # 计算真实类别与预测类别之间的差值，然后按照该差值调整回归系数
        error = classLabels[i] - h
        # 0.01*(1*1)*(1*n)
        print weights, "*"*10 , dataMatrix[i], "*"*10 , error
        weights = weights + alpha * error * dataMatrix[i]
    return weights
```

可以看到，随机梯度上升算法与梯度上升算法在代码上很相似，但也有一些区别: 第一，后者的变量 h 和误差 error 都是向量，而前者则全是数值；第二，前者没有矩阵的转换过程，所有变量的数据类型都是 NumPy 数组。

判断优化算法优劣的可靠方法是看它是否收敛，也就是说参数是否达到了稳定值，是否还会不断地变化？下图展示了随机梯度上升算法在 200 次迭代过程中回归系数的变化情况。其中的系数2，也就是 X2 只经过了 50 次迭代就达到了稳定值，但系数 1 和 0 则需要更多次的迭代。如下图所示:



针对这个问题，我们改进了之前的随机梯度上升算法，如下:

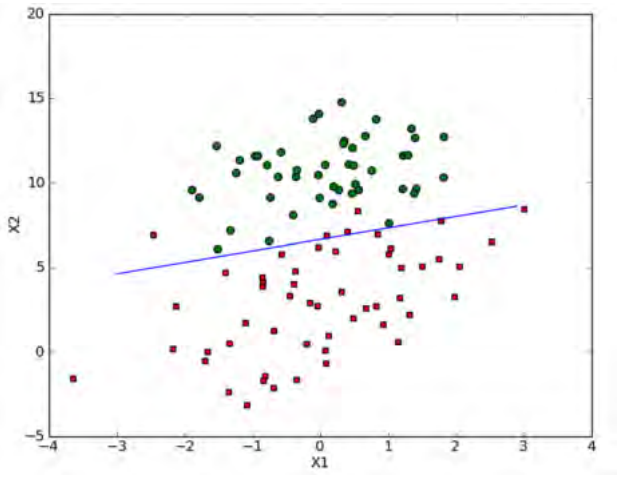
```
# 随机梯度上升算法（随机化）
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = shape(dataMatrix)
    weights = ones(n)   # 创建与列数相同的矩阵的系数矩阵，所有的元素都是1
    # 随机梯度，循环150,观察是否收敛
    for j in range(numIter):
        # [0, 1, 2 .. m-1]
        dataIndex = range(m)
        for i in range(m):
            # i和j的不断增大，导致alpha的值不断减少，但是不为0
            alpha = 4/(1.0+j+i)+0.0001    # alpha 会随着迭代不断减小，但永远不会减小到0，因为后边还有一个常数项0.0001
            # 随机产生一个 0～len()之间的一个值
            # random.uniform(x, y) 方法将随机生成下一个实数，它在[x,y]范围内,x是这个范围内的最小值，y是这个范围内的最大值。
            randIndex = int(random.uniform(0,len(dataIndex)))
            # sum(dataMatrix[i]*weights)为了求 f(x)的值， f(x)=a1*x1+b2*x2+..+nn*xn
            h = sigmoid(sum(dataMatrix[randIndex]*weights))
            error = classLabels[randIndex] - h
            # print weights, '__h=%s' % h, '__'*20, alpha, '__'*20, error, '__'*20, dataMatrix[randIndex]
            weights = weights + alpha * error * dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights
```

上面的改进版随机梯度上升算法，我们修改了两处代码。

第一处改进为 alpha 的值。alpha 在每次迭代的时候都会调整，这回缓解上面波动图的数据波动或者高频波动。另外，虽然 alpha 会随着迭代次数不断减少，但永远不会减小到 0，因为我们在计算公式中添加了一个常数项。

第二处修改为 randIndex 更新，这里通过随机选取样本拉来更新回归系数。这种方法将减少周期性的波动。这种方法每次随机从列表中选出一个值，然后从列表中删掉该值（再进行下一次迭代）。

程序运行之后能看到类似于下图的结果图。



完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/5.Logistic/logistic.py

项目案例2: 从疝气病症预测病马的死亡率

项目概述

使用 Logistic 回归来预测患有疝病的马的存活问题。疝病是描述马胃肠痛的术语。然而，这种病不一定源自马的胃肠问题，其他问题也可能引发马疝病。这个数据集中包含了医院检测马疝病的一些指标，有的指标比较主观，有的指标难以测量，例如马的疼痛级别。

开发流程

```
收集数据：给定数据文件
准备数据：用 Python 解析文本文件并填充缺失值
分析数据：可视化并观察数据
训练算法：使用优化算法，找到最佳的系数
测试算法：为了量化回归的效果，需要观察错误率。根据错误率决定是否回退到训练阶段，
        通过改变迭代的次数和步长的参数来得到更好的回归系数
使用算法：实现一个简单的命令行程序来手机马的症状并输出预测结果并非难事，
        这可以作为留给大家的一道习题
```

> 收集数据: 给定数据文件

病马的训练数据已经给出来了，如下形式存储在文本文件中:

```
1.000000    1.000000    39.200000    88.000000    20.000000    0.000000    0.000000    4.000000    1.000000
2.000000    1.000000    38.300000    40.000000    24.000000    1.000000    1.000000    3.000000    1.000000
```

> 准备数据: 用 Python 解析文本文件并填充缺失值

处理数据中的缺失值

假设有100个样本和20个特征，这些数据都是机器收集回来的。若机器上的某个传感器损坏导致一个特征无效时该怎么办？此时是否要扔掉整个数据？这种情况下，另外19个特征怎么办？ 它们是否还可以用？答案是肯定的。因为有时候数据相当昂贵，扔掉和重新获取都是不可取的，所以必须采用一些方法来解决这个问题。

下面给出了一些可选的做法：

- 使用可用特征的均值来填补缺失值；
- 使用特殊值来填补缺失值，如 -1；
- 忽略有缺失值的样本；
- 使用有相似样本的均值添补缺失值；
- 使用另外的机器学习算法预测缺失值。

现在，我们对下一节要用的数据集进行预处理，使其可以顺利地使用分类算法。在预处理需要做两件事:

- 所有的缺失值必须用一个实数值来替换，因为我们使用的 NumPy 数据类型不允许包含缺失值。我们这里选择实数 0 来替换所有缺失值，恰好能适用于 Logistic 回归。这样做的直觉在于，我们需要的是一个在更新时不会影响系数的值。回归系数的更新公式如下:

  weights = weights + alpha * error * dataMatrix[randIndex]

  如果 dataMatrix 的某个特征对应值为 0，那么该特征的系数将不做更新，即:

  weights = weights

  另外，由于 Sigmoid(0) = 0.5 ，即它对结果的预测不具有任何倾向性，因此我们上述做法也不会对误差造成任何影响。基于上述原因，将缺失值用 0 代替既可以保留现有数据，也不需要对优化算法进行修改。此外，该数据集中的特征取值一般不为 0，因此在某种意义上说它也满足 “特殊值” 这个要求。

- 如果在测试数据集中发现了一条数据的类别标签已经缺失，那么我们的简单做法是将该条数据丢弃。这是因为类别标签与特征不同，很难确定采用某个合适的值来替换。采用 Logistic 回归进行分类时这种做法是合理的，而如果采用类似 kNN 的方法就可能不太可行。

原始的数据集经过预处理后，保存成两个文件: horseColicTest.txt 和 horseColicTraining.txt 。

> 分析数据: 可视化并观察数据

将数据使用 MatPlotlib 打印出来，观察数据是否是我们想要的格式

> 训练算法: 使用优化算法，找到最佳的系数

下面给出 原始的梯度上升算法，随机梯度上升算法，改进版随机梯度上升算法 的代码:

```
# 正常的处理方案
# 两个参数：第一个参数==> dataMatIn 是一个2维NumPy数组，每列分别代表每个不同的特征，每行则代表每个训练样本。
# 第二个参数==> classLabels 是类别标签，它是一个 1*100 的行向量。为了便于矩阵计算，需要将该行向量转换为列向量，做法是将原向量转置，再将它赋值给
def gradAscent(dataMatIn, classLabels):
    # 转化为矩阵[[1,1,2],[1,1,2]....]
    dataMatrix = mat(dataMatIn)             # 转换为 NumPy 矩阵
    # 转化为矩阵[[0,1,0,1,0,1.....]]，并转制[[0],[1],[0].....]
    # transpose() 行列转置函数
    # 将行向量转化为列向量    =>  矩阵的转置
    labelMat = mat(classLabels).transpose() # 首先将数组转换为 NumPy 矩阵，然后再将行向量转置为列向量
    # m->数据量，样本数 n->特征数
    m,n = shape(dataMatrix)
    # print m, n, '__'*10, shape(dataMatrix.transpose()), '__'*100
    # alpha代表向目标移动的步长
    alpha = 0.001
    # 迭代次数
    maxCycles = 500
    # 生成一个长度和特征数相同的矩阵，此处n为3 -> [[1],[1],[1]]
    # weights 代表回归系数， 此处的 ones((n,1)) 创建一个长度和特征数相同的矩阵，其中的数全部都是 1
    weights = ones((n,1))
    for k in range(maxCycles):              #heavy on matrix operations
        # m*3 的矩阵 * 3*1 的单位矩阵 = m*1的矩阵
        # 那么乘上单位矩阵的意义，就代表：通过公式得到的理论值
        # 参考地址： 矩阵乘法的本质是什么？ https://www.zhihu.com/question/21351965/answer/31050145
        # print 'dataMatrix====', dataMatrix
        # print 'weights====', weights
        # n*3   *  3*1  = n*1
        h = sigmoid(dataMatrix*weights)     # 矩阵乘法
        # print 'hhhhhhh====', h
        # labelMat是实际值
        error = (labelMat - h)              # 向量相减
        # 0.001* (3*m)*(m*1) 表示在每一个列上的一个误差情况，最后得出 x1,x2,xn的系数的偏移量
        weights = weights + alpha * dataMatrix.transpose() * error # 矩阵乘法，最后得到回归系数
    return array(weights)


# 随机梯度上升
# 梯度上升优化算法在每次更新数据集时都需要遍历整个数据集，计算复杂都较高
```

```
# 随机梯度上升一次只用一个样本点来更新回归系数
def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.01
    # n*1的矩阵
    # 函数ones创建一个全1的数组
    weights = ones(n)   # 初始化长度为n的数组，元素全部为 1
    for i in range(m):
        # sum(dataMatrix[i]*weights)为了求 f(x)的值， f(x)=a1*x1+b2*x2+..+nn*xn,此处求出的 h 是一个具体的数值，而不是一个矩阵
        h = sigmoid(sum(dataMatrix[i]*weights))
        # print 'dataMatrix[i]===', dataMatrix[i]
        # 计算真实类别与预测类别之间的差值，然后按照该差值调整回归系数
        error = classLabels[i] - h
        # 0.01*(1*1)*(1*n)
        print weights, "*"*10 , dataMatrix[i], "*"*10 , error
        weights = weights + alpha * error * dataMatrix[i]
    return weights


# 随机梯度上升算法（随机化）
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = shape(dataMatrix)
    weights = ones(n)   # 创建与列数相同的矩阵的系数矩阵，所有的元素都是1
    # 随机梯度, 循环150,观察是否收敛
    for j in range(numIter):
        # [0, 1, 2 .. m-1]
        dataIndex = range(m)
        for i in range(m):
            # i和j的不断增大，导致alpha的值不断减少，但是不为0
            alpha = 4/(1.0+j+i)+0.0001    # alpha 会随着迭代不断减小，但永远不会减小到0，因为后边还有一个常数项0.0001
            # 随机产生一个 0～len()之间的一个值
            # random.uniform(x, y) 方法将随机生成下一个实数，它在[x,y]范围内,x是这个范围内的最小值，y是这个范围内的最大值。
            randIndex = int(random.uniform(0,len(dataIndex)))
            # sum(dataMatrix[i]*weights)为了求 f(x)的值， f(x)=a1*x1+b2*x2+..+nn*xn
            h = sigmoid(sum(dataMatrix[randIndex]*weights))
            error = classLabels[randIndex] - h
            # print weights, '__h=%s' % h, '__'*20, alpha, '__'*20, error, '__'*20, dataMatrix[randIndex]
            weights = weights + alpha * error * dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights
```

> 测试算法: 为了量化回归的效果，需要观察错误率。根据错误率决定是否回退到训练阶段，通过改变迭代的次数和步长的参数来得到更好的回归系数

Logistic 回归分类函数

```
# 分类函数，根据回归系数和特征向量来计算 Sigmoid的值
def classifyVector(inX, weights):
    '''
    Desc:
        最终的分类函数，根据回归系数和特征向量来计算 Sigmoid 的值，大于0.5函数返回1，否则返回0
    Args:
        inX -- 特征向量，features
        weights -- 根据梯度下降/随机梯度下降 计算得到的回归系数
    Returns:
        如果 prob 计算大于 0.5 函数返回 1
        否则返回 0
    '''
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5: return 1.0
    else: return 0.0

# 打开测试集和训练集,并对数据进行格式化处理
def colicTest():
    '''
    Desc:
        打开测试集和训练集，并对数据进行格式化处理
    Args:
        None
    Returns:
        errorRate -- 分类错误率
    '''
    frTrain = open('input/5.Logistic/horseColicTraining.txt')
    frTest = open('input/5.Logistic/horseColicTest.txt')
    trainingSet = []
    trainingLabels = []
    # 解析训练数据集中的数据特征和Labels
    # trainingSet 中存储训练数据集的特征，trainingLabels 存储训练数据集的样本对应的分类标签
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
```

```
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    # 使用 改进后的 随机梯度下降算法 求得在此数据集上的最佳回归系数 trainWeights
    trainWeights = stocGradAscent1(array(trainingSet), trainingLabels, 500)
    errorCount = 0
    numTestVec = 0.0
    # 读取 测试数据集 进行测试，计算分类错误的样本条数和最终的错误率
    for line in frTest.readlines():
        numTestVec += 1.0
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(array(lineArr), trainWeights)) != int(currLine[21]):
            errorCount += 1
    errorRate = (float(errorCount) / numTestVec)
    print "the error rate of this test is: %f" % errorRate
    return errorRate


# 调用 colicTest() 10次并求结果的平均值
def multiTest():
    numTests = 10
    errorSum = 0.0
    for k in range(numTests):
        errorSum += colicTest()
    print "after %d iterations the average error rate is: %f" % (numTests, errorSum/float(numTests))
```

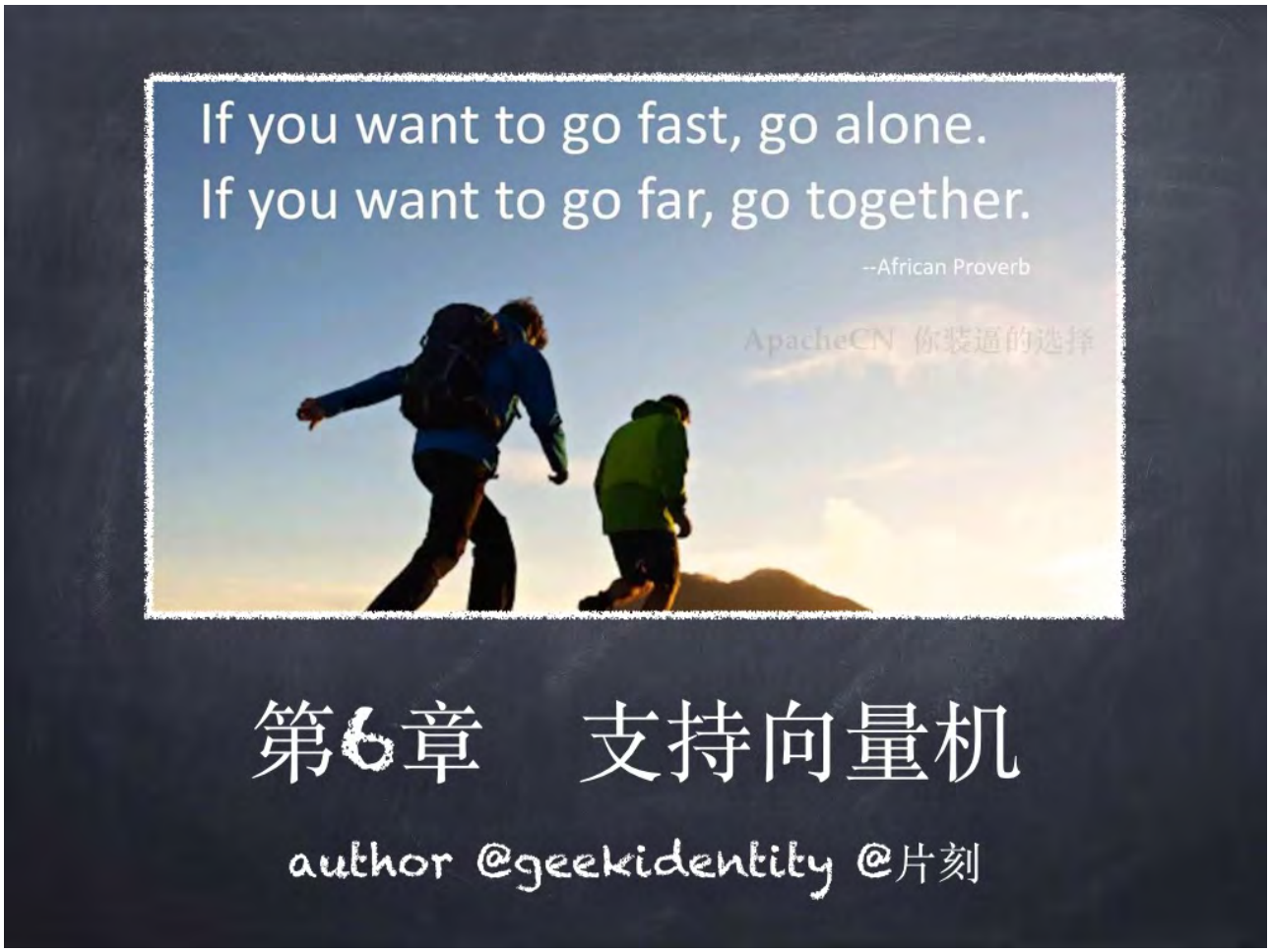
> 使用算法: 实现一个简单的命令行程序来手机马的症状并输出预测结果并非难事，这可以作为留给大家的一道习题

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/5.Logistic/logistic.py

---

- 作者：羊三 小瑶
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第6章 支持向量机



支持向量机 概述

支持向量机(Support Vector Machines, SVM)：是一种机器学习算法。

- 支持向量(Support Vector)就是离分隔超平面最近的那些点。
- 机(Machine)就是表示一种算法，而不是表示机器。

支持向量机 场景

- 要给左右两边的点进行分类
- 明显发现：选择D会比B、C分隔的效果要好很多。

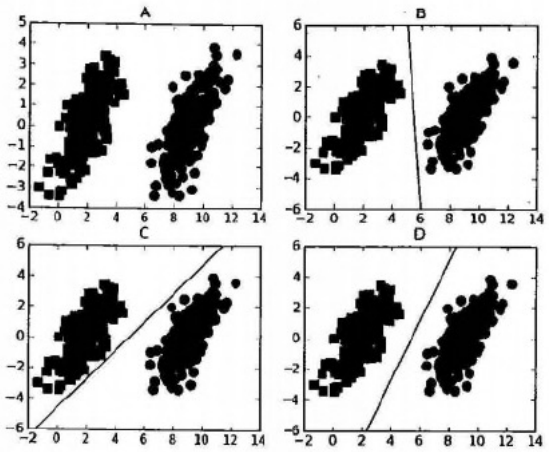
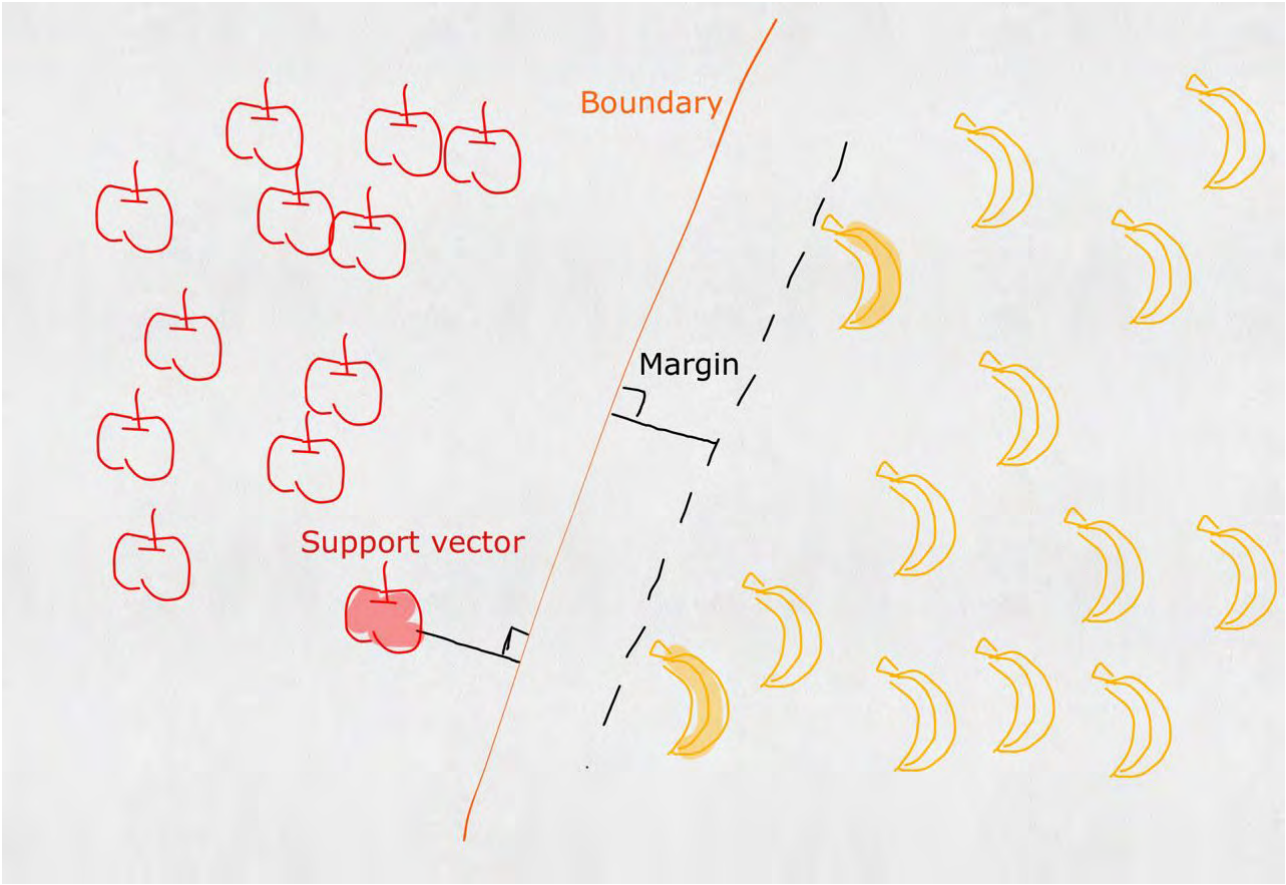


图6-2 A框中给出了一个线性可分的数据集，B、C、D框中各自给出了一条可以将两类数据分开的直线

支持向量机 原理

SVM 工作原理

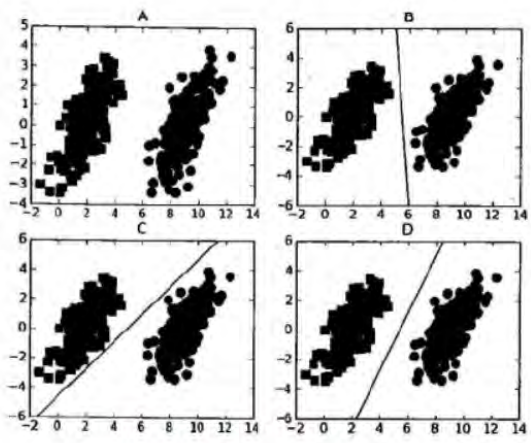


对于上述的苹果和香蕉，我们想象为2种水果类型的炸弹。（保证距离最近的炸弹，距离它们最远）

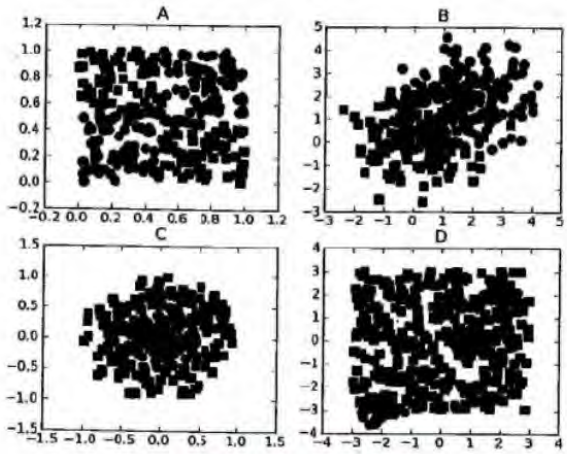
1. 寻找最大分类间距
2. 转而通过拉格朗日函数求优化的问题

- 数据可以通过画一条直线就可以将它们完全分开，这组数据叫`线性可分(linearly separable)`数据，而这条分隔直线称为`分隔超平面(separating hyperplane)`。

- 如果数据集上升到1024维呢？那么需要1023维来分隔数据集，也就说需要N-1维的对象来分隔，这个对象叫做超平面(hyperlane)，也就是分类的决策边界。



图A：线性可分



图B：线性不可分

寻找最大间隔

为什么寻找最大间隔

```
摘录地址：http://slideplayer.com/slide/8610144  (第12条信息)
Support Vector Machines: Slide 12 Copyright © 2001, 2003, Andrew W. Moore Why Maximum Margin?
denotes +1 denotes -1 f(x,w,b) = sign(w. x - b) The maximum margin linear classifier is the linear classifier with the, um, maximum margin.
This is the simplest kind of SVM (Called an LSVM) Support Vectors are those datapoints that the margin pushes up against

1.Intuitively this feels safest.
2.If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of ca
3.CV is easy since the model is immune to removal of any non-support-vector datapoints.
4.There's some theory that this is a good thing.
5.Empirically it works very very well.

* * *

1. 直觉上是安全的
2. 如果我们在边界的位置发生了一个小错误（它在垂直方向上被颠倒），这给我们最小的错误分类机会。
3. CV（Computer Vision 计算机视觉 - 这缩写看着可怕）很容易，因为该模型对任何非支持向量数据点的去除是免疫的。
4. 有一些理论，这是一件好事。
5. 通常它的工作非常好。
```

怎么寻找最大间隔

> 点到超平面的距离

- 分隔超平面函数间距: $y(x) = w^T x + b$
- 分类的结果： $f(x) = sign(w^T x + b)$ (sign表示>0为1，<0为-1，=0为0)
- 点到超平面的几何间距: $d(x) = (w^T x + b)/||w||$ (||w||表示w矩阵的二范式=> $\sqrt{w * w^T}$, 点到超平面的距离也是类似的)

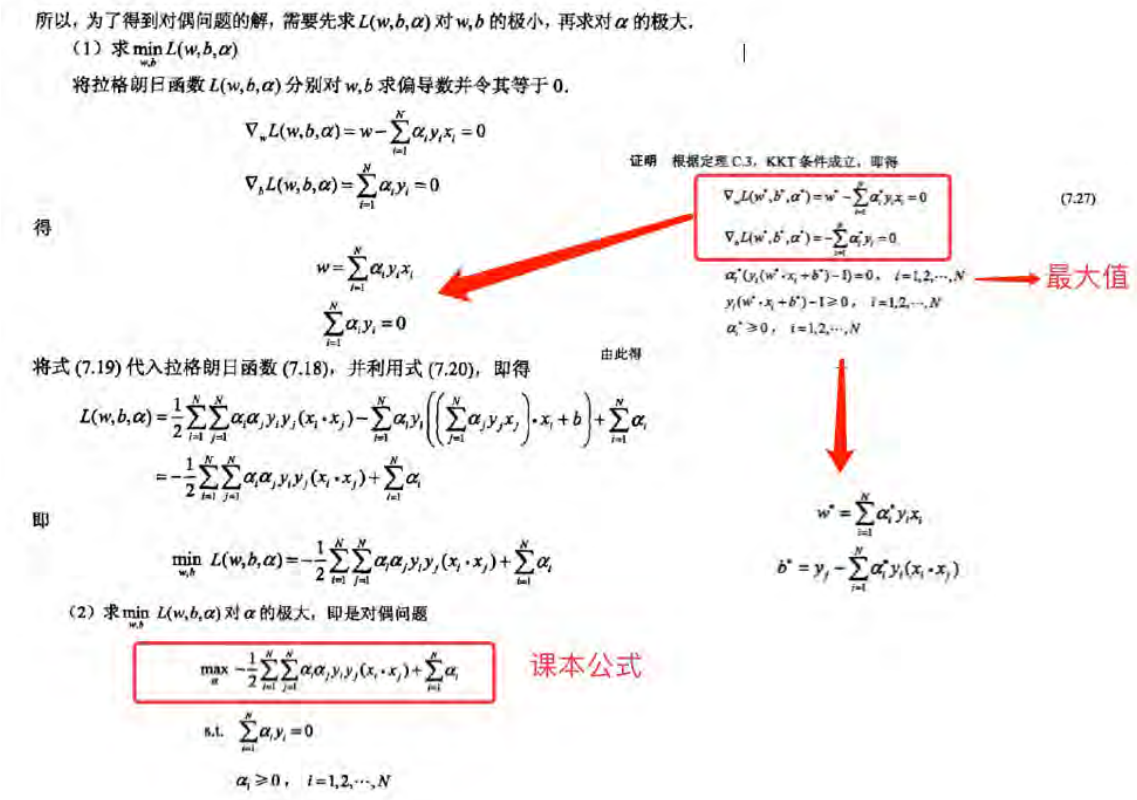
点到直线的距离公式

$$d = \left| \frac{Ax_0 + By_0 + C}{\sqrt{A^2 + B^2}} \right|$$

> 拉格朗日乘子法

- 类别标签用-1、1，是为了后期方便 $lable * (w^T x + b)$ 的标识和距离计算；如果 $lable * (w^T x + b) > 0$ 表示预测正确，否则预测错误。
- 现在目标很明确，就是要找到w和b，因此我们必须要找到最小间隔的数据点，也就是前面所说的支持向量。
  - 也就说，让最小的距离取最大.(最小的距离：就是最小间隔的数据点；最大：就是最大间距，为了找出最优超平面--最终就是支持向量)
  - 目标函数：$arg: max_{关于w,b}\left(min[lable * (w^T x + b)] * \frac{1}{||w||}\right)$
    1. 如果 $lable * (w^T x + b) > 0$ 表示预测正确，也称函数间隔，$||w||$ 可以理解为归一化，也称几何间隔。
    2. 令 $lable * (w^T x + b) >= 1$， 因为0~1之间，得到的点是存在误判的可能性，所以要保障 $min[lable * (w^T x + b)] = 1$，才能更好降低噪音数据影响。
    3. 所以本质上是求 $arg: max_{关于w,b}\frac{1}{||w||}$；也就说，我们约束(前提)条件是: $lable * (w^T x + b) = 1$
- 新的目标函数求解： $arg: max_{关于w,b}\frac{1}{||w||}$
  - => 就是求: $arg: min_{关于w,b}||w||$ (求矩阵会比较麻烦，如果x只是 $\frac{1}{2} * x^2$ 的偏导数，那么。。同样是求最小值)
  - => 就是求: $arg: min_{关于w,b}(\frac{1}{2} * ||w||^2)$ (二次函数求导，求极值，平方也方便计算)

    - 本质上就是求线性不等式的二次优化问题(求分隔超平面，等价于求解相应的凸二次规划问题)
- 通过拉格朗日乘子法，求二次优化问题
    - 假设需要求极值的目标函数 (objective function) 为 f(x,y)，限制条件为 φ(x,y)=M # M=1
    - 设g(x,y)=M-φ(x,y) # 临时φ(x,y)表示下文中 $label * (w^T x + b)$
    - 定义一个新函数: F(x,y,λ)=f(x,y)+λg(x,y)
    - a为λ（a>=0），代表要引入的拉格朗日乘子(Lagrange multiplier)
    - 那么： $L(w,b,\alpha) = \frac{1}{2} * ||w||^2 + \sum_{i=1}^{n} \alpha_i * [1 - label * (w^T x + b)]$
    - 因为：$label * (w^T x + b) >= 1, \alpha >= 0$, 所以 $\alpha * [1 - label * (w^T x + b)] <= 0$, $\sum_{i=1}^{n} \alpha_i * [1 - label * (w^T x + b)] <= 0$
    - 相当于求解： $max_{关于\alpha} L(w,b,\alpha) = \frac{1}{2} * ||w||^2$
    - 如果求： $min_{关于w,b} \frac{1}{2} * ||w||^2$，也就是要求： $min_{关于w,b}(max_{关于\alpha} L(w,b,\alpha))$
- 现在转化到对偶问题的求解
    - $min_{关于w,b}(max_{关于\alpha} L(w,b,\alpha))$ >= $max_{关于\alpha}(min_{关于w,b} L(w,b,\alpha))$
    - 现在分2步
    - 先求： $min_{关于w,b} L(w,b,\alpha) = \frac{1}{2} * ||w||^2 + \sum_{i=1}^{n} \alpha_i * [1 - label * (w^T x + b)]$
    - 就是求`L(w,b,a)`关于[w, b]的偏导数, 得到`w和b的值`，并化简为：`L和a的方程`。
    - 参考： 如果公式推导还是不懂，也可以参考《统计学习方法》李航-P103<学习的对偶算法>



- 终于得到课本上的公式： $max_{关于\alpha}\left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} label_i \cdot label_j \cdot \alpha_i \cdot \alpha_j \cdot < x_i, x_j >\right)$
- 约束条件： $a >= 0$ 并且 $\sum_{i=1}^{m} a_i \cdot label_i = 0$

> 松弛变量(slack variable)

- 我们知道几乎所有的数据都不那么干净, 通过引入松弛变量来`允许数据点可以处于分隔面错误的一侧`。
- 约束条件： $C >= a >= 0$并且 $\sum_{i=1}^{m} a_i \cdot label_i = 0$
- 这里常量C用于控制“最大化间隔”和“保证大部分点的函数间隔小于1.0” 这两个目标的权重。
- 常量C是一个常数，我们通过调节该参数得到不同的结果。一旦求出了所有的alpha，那么分隔超平面就可以通过这些alpha来表示。
- 这一结论十分直接，SVM中的主要工作就是要求解 alpha.

SMO 高效优化算法

- SVM有很多种实现，最流行的一种实现是： `序列最小优化(Sequential Minimal Optimization, SMO)算法`。
- 下面还会介绍一种称为 `核函数(kernel)` 的方式将SVM扩展到更多数据集上。
- 注意：`SVM几何含义比较直观，但其算法实现较复杂，牵扯大量数学公式的推导。`

> 序列最小优化(Sequential Minimal Optimization, SMO)

- 创建作者：John Platt
- 创建时间：1996年
- SMO用途：用于训练 SVM
- SMO目标：求出一系列 alpha 和 b,一旦求出 alpha，就很容易计算出权重向量 w 并得到分隔超平面。
- SMO思想：是将大优化问题分解为多个小优化问题来求解的。
- SMO原理：每次循环选择两个 alpha 进行优化处理，一旦找出一对合适的 alpha，那么就增大一个同时减少一个。
    - 这里指的合适必须要符合一定的条件

1. 这两个 alpha 必须要在间隔边界之外
2. 这两个 alpha 还没有进行过区间化处理或者不在边界上。

- 之所以要同时改变2个 alpha；原因是我们有一个约束条件： $\sum_{i=1}^{m} a_i \cdot label_i = 0$；如果只是修改一个 alpha，很可能导致约束条件失效。

> SMO 伪代码大致如下：

```
创建一个 alpha 向量并将其初始化为0向量
当迭代次数小于最大迭代次数时(外循环)
    对数据集中的每个数据向量(内循环):
        如果该数据向量可以被优化
            随机选择另外一个数据向量
            同时优化这两个向量
            如果两个向量都不能被优化，退出内循环
    如果所有向量都没被优化，增加迭代数目，继续下一次循环
```

SVM 开发流程

```
收集数据：可以使用任意方法。
准备数据：需要数值型数据。
分析数据：有助于可视化分隔超平面。
训练算法：SVM的大部分时间都源自训练，该过程主要实现两个参数的调优。
测试算法：十分简单的计算过程就可以实现。
使用算法：几乎所有分类问题都可以使用SVM，值得一提的是，SVM本身是一个二类分类器，对多类问题应用SVM需要对代码做一些修改。
```

SVM 算法特点

```
优点：泛化（由具体的、个别的扩大为一般的，就是说：模型训练完后的新样本）错误率低，计算开销不大，结果易理解。
缺点：对参数调节和核函数的选择敏感，原始分类器不加修改仅适合于处理二分类问题。
使用数据类型：数值型和标称型数据。
```

课本案例（无核函数）

项目概述

对小规模数据点进行分类

开发流程

> 收集数据

文本文件格式：

```
3.542485        1.977398        -1
3.018896        2.556416        -1
7.551510        -1.580030       1
2.114999        -0.004466       -1
8.127113        1.274372        1
```

> 准备数据

```
def loadDataSet(fileName):
    """
    对文件进行逐行解析，从而得到第行的类标签和整个特征矩阵
    Args:
        fileName 文件名
    Returns:
        dataMat  特征矩阵
        labelMat 类标签
    """
    dataMat = []
    labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat, labelMat
```

> 分析数据: 无

> 训练算法

```
def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    """smoSimple

    Args:
        dataMatIn    特征集合
        classLabels  类别标签
```

```
    C   松弛变量(常量值)，允许有些数据点可以处于分隔面的错误一侧。
        控制最大化间隔和保证大部分的函数间隔小于1.0这两个目标的权重。
        可以通过调节该参数达到不同的结果。
    toler   容错率（是指在某个体系中能减小一些因素或选择对某个系统产生不稳定的概率。）
    maxIter 退出前最大的循环次数
Returns:
    b       模型的常量值
    alphas  拉格朗日乘子
"""
dataMatrix = mat(dataMatIn)
# 矩阵转置 和 .T 一样的功能
labelMat = mat(classLabels).transpose()
m, n = shape(dataMatrix)

# 初始化 b和alphas(alpha有点类似权重值。)
b = 0
alphas = mat(zeros((m, 1)))

# 没有任何alpha改变的情况下遍历数据的次数
iter = 0
while (iter < maxIter):
    # w = calcWs(alphas, dataMatIn, classLabels)
    # print("w:", w)

    # 记录alpha是否已经进行优化，每次循环时设为0，然后再对整个集合顺序遍历
    alphaPairsChanged = 0
    for i in range(m):
        # print 'alphas=', alphas
        # print 'labelMat=', labelMat
        # print 'multiply(alphas, labelMat)=', multiply(alphas, labelMat)
        # 我们预测的类别 y[i] = w^Tx[i]+b; 其中因为 w = Σ(1~n) a[n]*lable[n]*x[n]
        fXi = float(multiply(alphas, labelMat).T*(dataMatrix*dataMatrix[i, :].T)) + b
        # 预测结果与真实结果比对，计算误差Ei
        Ei = fXi - float(labelMat[i])

        # 约束条件 (KKT条件是解决最优化问题的时用到的一种方法。我们这里提到的最优化问题通常是指对于给定的某一函数，求其在指定作用域上的全局最
        # 0<=alphas[i]<=C，但由于0和C是边界值，我们无法进行优化，因为需要增加一个alphas和降低一个alphas。
        # 表示发生错误的概率：labelMat[i]*Ei 如果超出了 toler， 才需要优化。至于正负号，我们考虑绝对值就对了。
        '''
        # 检验训练样本(xi, yi)是否满足KKT条件
        yi*f(i) >= 1 and alpha = 0 (outside the boundary)
        yi*f(i) == 1 and 0<alpha< C (on the boundary)
        yi*f(i) <= 1 and alpha = C (between the boundary)
        '''
        if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or ((labelMat[i]*Ei > toler) and (alphas[i] > 0)):

            # 如果满足优化的条件，我们就随机选取非i的一个点，进行优化比较
            j = selectJrand(i, m)
            # 预测j的结果
            fXj = float(multiply(alphas, labelMat).T*(dataMatrix*dataMatrix[j, :].T)) + b
            Ej = fXj - float(labelMat[j])
            alphaIold = alphas[i].copy()
            alphaJold = alphas[j].copy()

            # L和H用于将alphas[j]调整到0-C之间。如果L==H，就不做任何改变，直接执行continue语句
            # labelMat[i] != labelMat[j] 表示异侧，就相减，否则是同侧，就相加。
            if (labelMat[i] != labelMat[j]):
                L = max(0, alphas[j] - alphas[i])
                H = min(C, C + alphas[j] - alphas[i])
            else:
                L = max(0, alphas[j] + alphas[i] - C)
                H = min(C, alphas[j] + alphas[i])
            # 如果相同，就没发优化了
            if L == H:
                print("L==H")
                continue

            # eta是alphas[j]的最优修改量，如果eta==0，需要退出for循环的当前迭代过程
            # 参考《统计学习方法》李航-P125~P128<序列最小最优化算法>
            eta = 2.0 * dataMatrix[i, :]*dataMatrix[j, :].T - dataMatrix[i, :]*dataMatrix[i, :].T - dataMatrix[j, :]*dataMatrix[j
            if eta >= 0:
                print("eta>=0")
                continue

            # 计算出一个新的alphas[j]值
            alphas[j] -= labelMat[j]*(Ei - Ej)/eta
            # 并使用辅助函数，以及L和H对其进行调整
            alphas[j] = clipAlpha(alphas[j], H, L)
            # 检查alpha[j]是否只是轻微的改变，如果是的话，就退出for循环。
            if (abs(alphas[j] - alphaJold) < 0.00001):
                print("j not moving enough")
                continue
```

```
                # 然后alphas[i]和alphas[j]同样进行改变，虽然改变的大小一样，但是改变的方向正好相反
                alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])
                # 在对alpha[i], alpha[j] 进行优化之后，给这两个alpha值设置一个常数b。
                # w= Σ[1~n] ai*yi*xi => b = yj- Σ[1~n] ai*yi(xi*xj)
                # 所以：  b1 - b = (y1-y) - Σ[1~n] yi*(a1-a)*(xi*x1)
                # 为什么减2遍？ 因为是 减去Σ[1~n]，正好2个变量i和j，所以减2遍
                b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i, :]*dataMatrix[i, :].T - labelMat[j]*(alphas[j]-alphaJold
                b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i, :]*dataMatrix[j, :].T - labelMat[j]*(alphas[j]-alphaJold
                if (0 < alphas[i]) and (C > alphas[i]):
                    b = b1
                elif (0 < alphas[j]) and (C > alphas[j]):
                    b = b2
                else:
                    b = (b1 + b2)/2.0
                alphaPairsChanged += 1
                print("iter: %d i:%d, pairs changed %d" % (iter, i, alphaPairsChanged))
        # 在for循环外，检查alpha值是否做了更新，如果在更新则将iter设为0后继续运行程序
        # 知道更新完毕后，iter次循环无变化，才推出循环。
        if (alphaPairsChanged == 0):
            iter += 1
        else:
            iter = 0
        print("iteration number: %d" % iter)
    return b, alphas
```

完整代码地址：SVM简化版，应用简化版SMO算法处理小规模数据集: https://github.com/apachecn/MachineLearning/blob/master/src/python/6.SVM/svm-simple.py

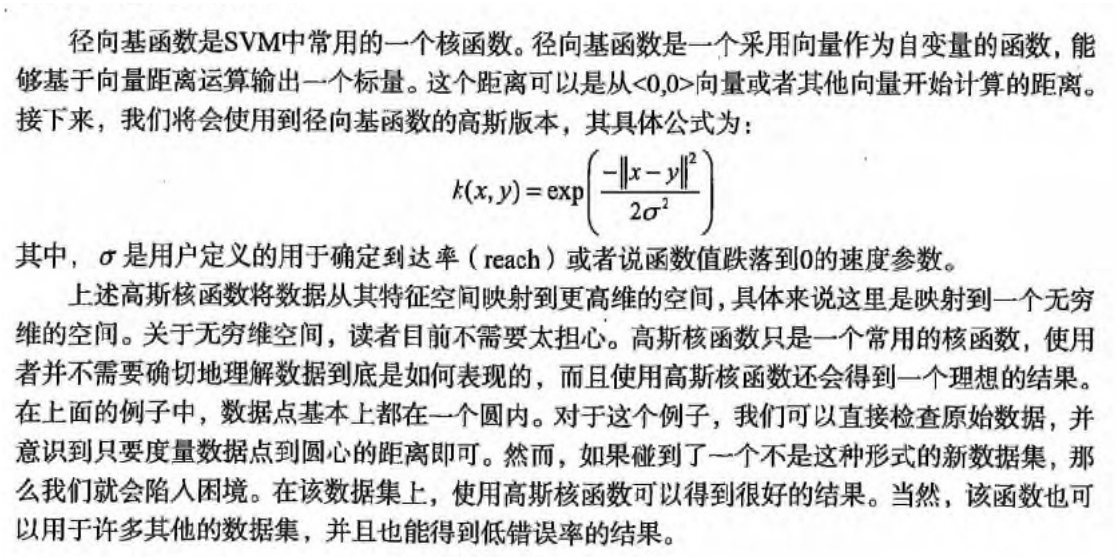
完整代码地址：SVM完整版，使用完整 Platt SMO算法加速优化，优化点：选择alpha的方式不同: https://github.com/apachecn/MachineLearning/blob/master/src/python/6.SVM/svm-complete_Non-Kernel.py

核函数(kernel) 使用

- 对于线性可分的情况，效果明显
- 对于非线性的情况也一样，此时需要用到一种叫`核函数(kernel)`的工具将数据转化为分类器易于理解的形式。

> 利用核函数将数据映射到高维空间

- 使用核函数：可以将数据从某个特征空间到另一个特征空间的映射。（通常情况下：这种映射会将低维特征空间映射到高维空间。）
- 如果觉得特征空间很装逼、很难理解。
- 可以把核函数想象成一个包装器(wrapper)或者是接口(interface)，它能将数据从某个很难处理的形式转换成为另一个较容易处理的形式。
- 经过空间转换后：低维需要解决的非线性问题，就变成了高维需要解决的线性问题。
- SVM 优化特别好的地方，在于所有的运算都可以写成内积(inner product: 是指2个向量相乘，得到单个标量 或者 数值)；内积替换成核函数的方式被称为`核技巧(kernel trick)`或者`核"变电"(kernel substation)`
- 核函数并不仅仅应用于支持向量机，很多其他的机器学习算法也都用到核函数。最流行的核函数：径向基函数(radial basis function)
- 径向基函数的高斯版本，其具体的公式为：



径向基函数是SVM中常用的一个核函数。径向基函数是一个采用向量作为自变量的函数，能够基于向量距离运算输出一个标量。这个距离可以是从<0,0>向量或者其他向量开始计算的距离。接下来，我们将会使用到径向基函数的高斯版本，其具体公式为：

$$k(x,y)=\exp\left(\frac{-\|x-y\|^2}{2\sigma^2}\right)$$

其中，σ是用户定义的用于确定到达率（reach）或者说函数值跌落到0的速度参数。

上述高斯核函数将数据从其特征空间映射到更高维的空间，具体来说这里是映射到一个无穷维的空间。关于无穷维空间，读者目前不需要太担心。高斯核函数只是一个常用的核函数，使用者并不需要确切地理解数据到底是如何表现的，而且使用高斯核函数还会得到一个理想的结果。在上面的例子中，数据点基本上都在一个圆内。对于这个例子，我们可以直接检查原始数据，并意识到只要度量数据点到圆心的距离即可。然而，如果碰到了一个不是这种形式的新数据集，那么我们就会陷入困境。在该数据集上，使用高斯核函数可以得到很好的结果。当然，该函数也可以用于许多其他的数据集，并且也能得到低错误率的结果。

项目案例: 手写数字识别的优化（有核函数）

项目概述

> 你的老板要求：你写的那个手写识别程序非常好，但是它占用内存太大。顾客无法通过无线的方式下载我们的应用。
> 所以：我们可以考虑使用支持向量机，保留支持向量就行（knn需要保留所有的向量），就可以获得非常好的效果。

开发流程

> 收集数据：提供的文本文件

```
00000000000000001111000000000000
00000000000000011111111000000000
00000000000000011111111100000000
00000000000000011111111110000000
00000000000000011111111110000000
00000000000000111111111100000000
00000000000000111111111100000000
00000000000000111111111100000000
00000000000000111111111100000000
00000000000000111111111100000000
00000000000000111111111000000000
00000000000001111111111000000000
00000000000011111111111000000000
00000000000111111111110000000000
00000000001111111111111000000000
00000001111111111111111000000000
00000011111111111111110000000000
00000111111111111111110000000000
00000111111111111111110000000000
00000011111111111111110000000000
00000011111110111111110000000000
00000001111100111111110000000000
00000000000000011111110000000000
00000000000000011111100000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000111111110000000000
00000000000000001111111100000000
00000000000000001111111100000000
```

> 准备数据：基于二值图像构造向量

将 32*32的文本转化为 1*1024的矩阵

```
def img2vector(filename):
    returnVect = zeros((1, 1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0, 32 * i + j] = int(lineStr[j])
    return returnVect

def loadImages(dirName):
    from os import listdir
    hwLabels = []
    print(dirName)
    trainingFileList = listdir(dirName)  # load the training set
    m = len(trainingFileList)
    trainingMat = zeros((m, 1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]  # take off .txt
        classNumStr = int(fileStr.split('_')[0])
        if classNumStr == 9:
            hwLabels.append(-1)
        else:
            hwLabels.append(1)
        trainingMat[i, :] = img2vector('%s/%s' % (dirName, fileNameStr))
    return trainingMat, hwLabels
```

> 分析数据：对图像向量进行目测

> 训练算法：采用两种不同的核函数，并对径向基核函数采用不同的设置来运行SMO算法

```
def kernelTrans(X, A, kTup):  # calc the kernel or transform data to a higher dimensional space
    """
    核转换函数
    Args:
        X     dataMatIn数据集
        A     dataMatIn数据集的第i行的数据
        kTup  核函数的信息

    Returns:

    """
```

```
    m, n = shape(X)
    K = mat(zeros((m, 1)))
    if kTup[0] == 'lin':
        # linear kernel:   m*n * n*1 = m*1
        K = X * A.T
    elif kTup[0] == 'rbf':
        for j in range(m):
            deltaRow = X[j, :] - A
            K[j] = deltaRow * deltaRow.T
        # 径向基函数的高斯版本
        K = exp(K / (-1 * kTup[1] ** 2))  # divide in NumPy is element-wise not matrix like Matlab
    else:
        raise NameError('Houston We Have a Problem -- That Kernel is not recognized')
    return K

def smoP(dataMatIn, classLabels, C, toler, maxIter, kTup=('lin', 0)):
    """
    完整SMO算法外循环，与smoSimple有些类似，但这里的循环退出条件更多一些
    Args:
        dataMatIn    数据集
        classLabels  类别标签
        C    松弛变量(常量值)，允许有些数据点可以处于分隔面的错误一侧。
            控制最大化间隔和保证大部分的函数间隔小于1.0这两个目标的权重。
            可以通过调节该参数达到不同的结果。
        toler   容错率
        maxIter 退出前最大的循环次数
        kTup    包含核函数信息的元组
    Returns:
        b       模型的常量值
        alphas  拉格朗日乘子
    """

    # 创建一个 optStruct 对象
    oS = optStruct(mat(dataMatIn), mat(classLabels).transpose(), C, toler, kTup)
    iter = 0
    entireSet = True
    alphaPairsChanged = 0

    # 循环遍历：循环maxIter次 并且 （alphaPairsChanged存在可以改变 or 所有行遍历一遍）
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0

        #  当entireSet=true or 非边界alpha对没有了；就开始寻找 alpha对，然后决定是否要进行else。
        if entireSet:
            # 在数据集上遍历所有可能的alpha
            for i in range(oS.m):
                # 是否存在alpha对，存在就+1
                alphaPairsChanged += innerL(i, oS)
                # print("fullSet, iter: %d i:%d, pairs changed %d" % (iter, i, alphaPairsChanged))
            iter += 1

        # 对已存在 alpha对，选出非边界的alpha值，进行优化。
        else:
            # 遍历所有的非边界alpha值，也就是不在边界0或C上的值。
            nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i, oS)
                # print("non-bound, iter: %d i:%d, pairs changed %d" % (iter, i, alphaPairsChanged))
            iter += 1

        # 如果找到alpha对，就优化非边界alpha值，否则，就重新进行寻找，如果寻找一遍 遍历所有的行还是没找到，就退出循环。
        if entireSet:
            entireSet = False  # toggle entire set loop
        elif (alphaPairsChanged == 0):
            entireSet = True
        print("iteration number: %d" % iter)
    return oS.b, oS.alphas
```

> 测试算法：便携一个函数来测试不同的和函数并计算错误率

```
def testDigits(kTup=('rbf', 10)):

    # 1. 导入训练数据
    dataArr, labelArr = loadImages('input/6.SVM/trainingDigits')
    b, alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, kTup)
    datMat = mat(dataArr)
    labelMat = mat(labelArr).transpose()
    svInd = nonzero(alphas.A > 0)[0]
    sVs = datMat[svInd]
    labelSV = labelMat[svInd]
    # print("there are %d Support Vectors" % shape(sVs)[0])
    m, n = shape(datMat)
```

```
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs, datMat[i, :], kTup)
        # 1*m * m*1 = 1*1 单个预测结果
        predict = kernelEval.T * multiply(labelSV, alphas[svInd]) + b
        if sign(predict) != sign(labelArr[i]): errorCount += 1
    print("the training error rate is: %f" % (float(errorCount) / m))

    # 2. 导入测试数据
    dataArr, labelArr = loadImages('input/6.SVM/testDigits')
    errorCount = 0
    datMat = mat(dataArr)
    labelMat = mat(labelArr).transpose()
    m, n = shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs, datMat[i, :], kTup)
        # 1*m * m*1 = 1*1 单个预测结果
        predict = kernelEval.T * multiply(labelSV, alphas[svInd]) + b
        if sign(predict) != sign(labelArr[i]): errorCount += 1
    print("the test error rate is: %f" % (float(errorCount) / m))
```

> 使用算法：一个图像识别的完整应用还需要一些图像处理的知识，这里并不打算深入介绍

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/6.SVM/svm-complete.py

---

- 作者：片刻 geekidentity
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

SVM

> 声明

```
阅读本文前，需要您懂一些高等数学、概率论、线性代数的相关知识，以便更好理解。
```

```
下面这些关于 SVM 的理解，是根据知乎和其他博客或者网站中查询到的资料加以整理，
并结合 ApacheCN 这段时间的撸代码和相关研究得到，有理解有误的地方还望大家指出，谢谢。
再次感谢网上的大佬们的无私贡献。

ApacheCN: http://www.apachecn.org/
ApacheCN MachineLearning github: https://github.com/apachecn/MachineLearning

网上资料参考链接:https://www.zhihu.com/question/21094489
            http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
            https://zhuanlan.zhihu.com/p/26891427?utm_medium=social&utm_source=qq
            https://zhuanlan.zhihu.com/p/21308801?utm_medium=social&utm_source=qq
            http://blog.csdn.net/v_july_v/article/details/7624837
```

Overview

What's the SVM?

^_^ 首先，支持向量机不是一种机器，而是一种机器学习算法。

1、SVM - Support Vector Machine ，俗称支持向量机，是一种 supervised learning （监督学习）算法，属于 classification （分类）的范畴。

2、在数据挖掘的应用中，与 unsupervised learning （无监督学习）的 Clustering（聚类）相对应和区别。

3、广泛应用于 Machine Learning （机器学习），Computer Vision （计算机视觉，装逼一点说，就是 cv）和 Data Mining （数据挖掘）当中。

“ Machine （机）” 是什么？

Classification Machine，是分类器，这个没什么好说的。也可以理解为算法，机器学习领域里面常常用 “机” 也就是 machine 这个字表示算法。

“支持向量” 又是什么？

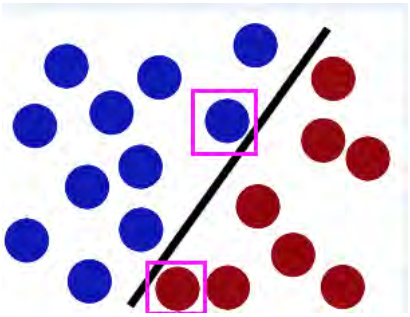
通俗理解： support vector （支持向量）的意思就是 数据集中的某些点，位置比较特殊。比如 x+y-2=0 这条直线，直线上面区域 x+y-2>0 的全是 A 类，下面的 x+y-2<0 的全是 B 类，我们找这条直线的时候，一般就看聚集在一起的两类数据，他们各自的 最边缘 位置的点，也就是最靠近划分直线的那几个点，而其他点对这条直线的最终位置的确定起不了作用，所以我姑且叫这些点叫 “支持点”（意思就是有用的点），但是在数学上，没这种说法，数学里的点，又可以叫向量，比如 二维点 (x,y) 就是二维向量，三维度的就是三维向量 (x,y,z)。所以 “支持点” 改叫 “支持向量” ，听起来比较专业，而且又装逼，何乐而不为呢？是吧...

不通俗的理解： 在 maximum margin （最大间隔）上的这些点就叫 “支持向量”，我想补充的是为啥这些点就叫 “支持向量” ，因为最后的 classification machine （分类器）的表达式里只含用这些 “支持向量” 的信息，而与其他数据点无关：

$$\mathbf{w} \cdot \varphi(\mathbf{x}) = \sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})$$

在这个表达式中，只有支持向量的系数 α_i 不等于 0 。

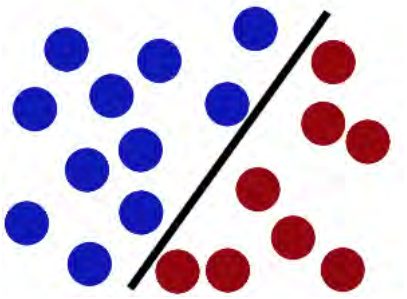
如果还是不怎么理解，不要紧，看下图:



“支持向量” 就是图中用紫色框框圈出来的点...

Concept （相关概念）

我们先看一张图



`linearly separable （线性可分）`: 如上图中的两组数据，它们之间已经分的足够开了，因此很容易就可以在图中画出一条直线将两组数据点分开。在这种情况下，这组数据就被称为线性可分数据。

`separating hyperplane（分隔超平面）`: 上述将数据集分隔开来的直线称为分隔超平面。

`hyperplane（超平面）`: 在上面给出的例子中，由于数据点都在二维平面上，所以此时分隔超平面就只是一条直线。但是，如果所给的数据集是三维的，那么此时用来分隔数据的就是一个平面。显而易见，更高纬度的情况可以依此类推。如果数据是 1024 维的，那么就需要一个 1023 维的某某对象（不是你们的男（女）票）来对数据进行分隔。这个 1023 维的某某对象到底应该叫什么呢？ N-1 维呢？该对象被称为超平面，也就是分类的决策边界。分布在超平面一侧的所有数据都属于某个类别，而分布在另一侧的所有数据则属于另一个类别。

`margin（间隔）`: 我们希望能通过上述的方式来构建分类器，即如果数据点离决策边界越远，那么其最后的预测结果也就越可信。既然这样，我们希望找到离分隔超平面最近的点，确保它们离分隔面的距离尽可能远。这里所说的点到分隔面的距离就是 间隔。我们希望间隔尽可能地大，这是因为如果我们犯错或者在有限数据上训练分类器的话，我们希望分类器尽可能健壮。

`支持向量（support vector）`: 就是上面所说的离分隔超平面最近的那些点。

`分类器`: 分类器就是给定一个样本的数据，判定这个样本属于哪个类别的算法。例如在股票涨跌预测中，我们认为前一天的交易量和收盘价对于第二天的涨跌是有影响的，那么分类器就是通过样本的交易量和收盘价预测第二天的涨跌情况的算法。

`特征`: 在分类问题中，输入到分类器中的数据叫做特征。以上面的股票涨跌预测问题为例，特征就是前一天的交易量和收盘价。

`线性分类器`: 线性分类器是分类器中的一种，就是判定分类结果的根据是通过特征的线性组合得到的，不能通过特征的非线性运算结果作为判定根据。还以上面的股票涨跌预测问题为例，判断的依据只能是前一天的交易量和收盘价的线性组合，不能将交易量和收盘价进行开方，平方等运算。

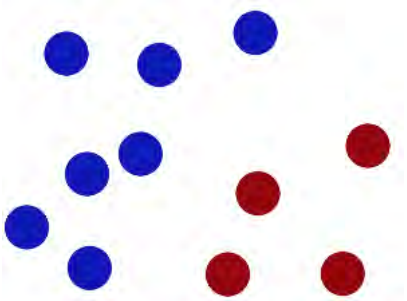
How does it work? （SVM 原理）

1、引用知乎上 @简之 大佬的回答：

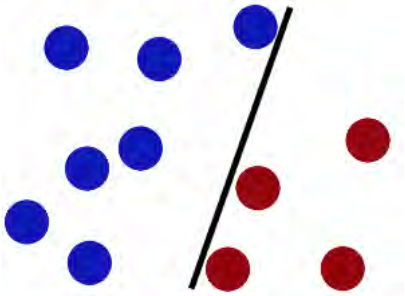
首先我们讲个故事:

在很久以前的情人节，大侠要去救他的爱人，但魔鬼和他玩了一个游戏。

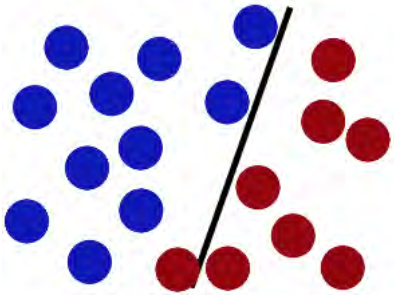
魔鬼在桌子上似乎有规律放了两种颜色的球，说：“你用一根棍分开它们？要求：尽量在放更多球之后，仍然适用。”



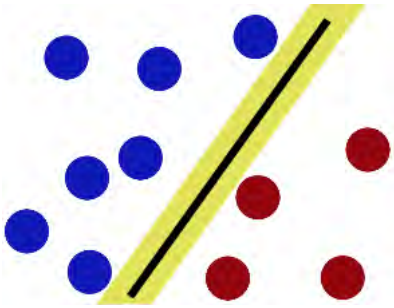
于是大侠这样放，干的不错？



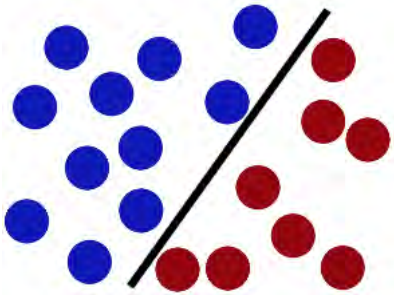
然后魔鬼，又在桌上放了更多的球，似乎有一个球站错了阵营。



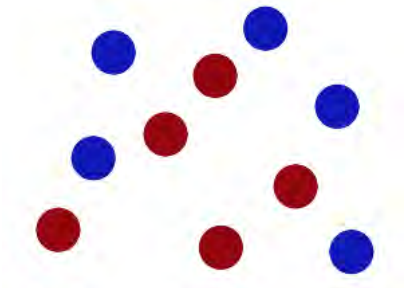
SVM 就是试图把棍放在最佳位置，好让在棍的两边有尽可能大的间隙。



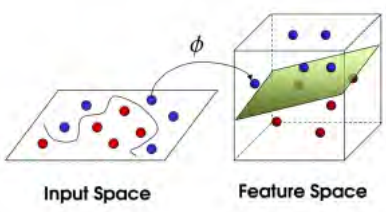
现在即使魔鬼放了更多的球，棍仍然是一个好的分界线。



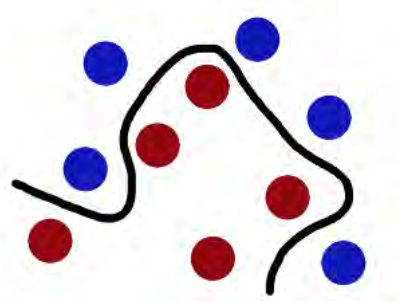
然后，在 SVM 工具箱中有另一个更加重要的 trick。 魔鬼看到大侠已经学会了一个 trick ，于是魔鬼给了大侠一个新的挑战。



现在，大侠没有棍可以很好帮他分开两种球了，现在怎么办呢？当然像所有武侠片中一样大侠桌子一拍，球飞到空中。然后，凭借大侠的轻功，大侠抓起一张纸，插到了两种球的中间。



现在，从魔鬼的角度看这些球，这些球看起来像是被一条曲线分开了。



再之后，无聊的大人们，把这些球叫做 「data」，把棍子叫做 「classifier 」，最大间隙 trick 叫做「optimization 」， 拍桌子叫做「kernelling 」，那张纸叫做「hyperplane 」 。

有梯子的童鞋，可以看一下这个地方，看视频来更直观的感受：

https://www.youtube.com/watch?v=3liCbRZPrZA

2、引用知乎 @开膛手水货 大佬的回答，我认为是超级通俗的一个版本：

支持向量机是用来解决分类问题的。

先考虑最简单的情况，豌豆和米粒，用晒子很快可以分开，小颗粒漏下去，大颗粒保留。

用一个函数来表示就是当直径 d 大于某个值 D ，就判定为豌豆，小于某个值就是米粒。

d>D, 豌豆

d<D,米粒

在数轴上就是在d左边就是米粒，右边就是绿豆，这是一维的情况。

但是实际问题没这么简单，考虑的问题不单单是尺寸，一个花的两个品种，怎么分类？

假设决定他们分类的有两个属性，花瓣尺寸和颜色。单独用一个属性来分类，像刚才分米粒那样，就不行了。这个时候我们设置两个值 尺寸 x 和颜色 y.

我们把所有的数据都丢到 x-y 平面上作为点，按道理如果只有这两个属性决定了两个品种，数据肯定会按两类聚集在这个二维平面上。

我们只要找到一条直线，把这两类划分开来，分类就很容易了，以后遇到一个数据，就丢进这个平面，看在直线的哪一边，就是哪一类。

比如 x+y-2=0 这条直线，我们把数据 (x,y) 代入，只要认为 x+y-2>0 的就是 A 类， x+y-2<0 的就是 B 类。

以此类推，还有三维的，四维的，N维的 属性的分类，这样构造的也许就不是直线，而是平面，超平面。

一个三维的函数分类 ：x+y+z-2=0，这就是个分类的平面了。

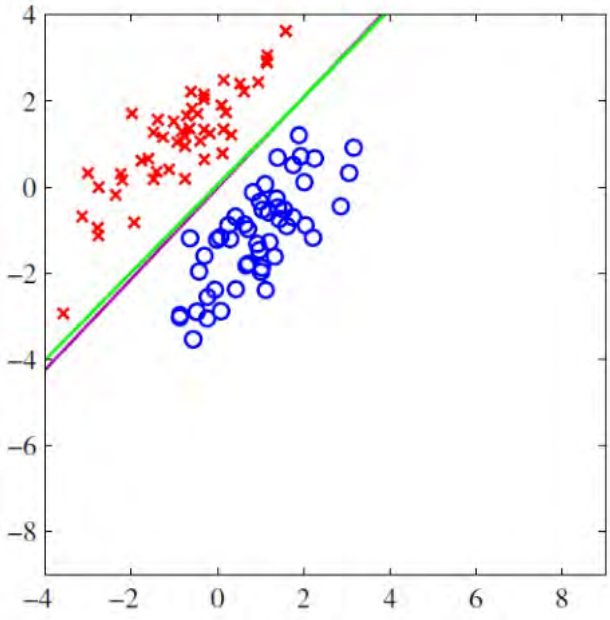
有时候，分类的那条线不一定是直线，还有可能是曲线，我们通过某些函数来转换，就可以转化成刚才的哪种多维的分类问题，这个就是核函数的思想。

例如：分类的函数是个圆形 x^2+y^2-4=0 。这个时候令 x^2=a ; y^2=b ,还不就变成了a+b-4=0 这种直线问题了。

这就是支持向量机的思想。

3、引用 @胡KF 大佬的回答（这个需要一些数学知识）：

如图的例子，（训练集）红色点是我们已知的分类1，（训练集）蓝色点是已知的分类2，我们想寻找一个分界超平面（图中绿线）（因为示例是二维数据点，所以只是一条线，如果数据是三维的就是平面，如果是三维以上就是超平面）把这两类完全分开，这样的话再来一个样本点需要我们预测的话，我们就可以根据这个分界超平面预测出分类结果。



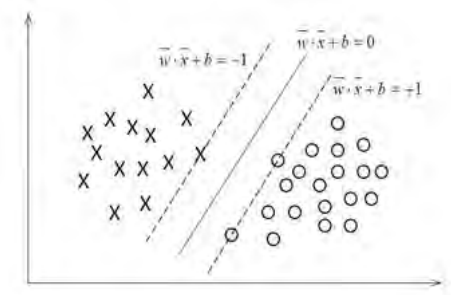
那我们如何选择这个分类超平面呢？从数学上说，超平面的公式是,也就是说如何选取这个 ω（是个向量）。

传统方法是根据最小二乘错误法（least squared error），首先随便定选取一个随机平面，也就是随机选取 ω 和 b，然后想必会在训练集中产生大量的错误分类，也就是说，$\omega^T x + b$ 结果应该大于 0 的时候小于 0 ，应该小于 0 的时候大于 0 。这时候有一个错误损失，也就是说对于所有错误的分类，他们的平方和（least squared error）为: $\sum_i^n (\omega^T x_i + b)^2$，最小二乘法的目标就是让这个值趋于最小，对 ω 求导取 0 ，采用梯度下降算法，可以求出错误平方和的极值，求出最优的 ω ，也就是求出最优的超平面。（可以证明，如果基函数是指数族函数，求出的超平面是全局最优的）。

那我们 SVM 算法的思路是怎样的呢？

不同于传统的最小二乘策略的思想，我们采用一种新的思路，这个分界面有什么样的特征呢？

第一，它 “夹” 在两类样本点之间；第二，它离两类样本点中所有 “离它最近的点” ，都离它尽可能的远。如图所示：



在虚线上的点，就是我们所找到的离分解超平面最近的样本点，X 类中找到了一个，O 类找到了两个。我们需要分类超平面离这三个样本点都尽可能的远，也就是说，它处在两条虚线的中间。这就是我们找到的分界超平面。

另外，这里我们就可以解释什么是 “支持向量” 了，支持向量就是虚线上的离分类超平面最近的样本点，因为每一个样本点都是一个多维的向量，向量的每一个维度都是这个样本点的一个特征。比如在根据身高，体重，特征进行男女分类的时候，每一个人是一个向量，向量有两个维度，第一维是身高，第二维是体重。

介绍完 SVM 的基本思想，我们来探讨一下如何用数学的方法进行 SVM 分类。

首先我们需要把刚刚说的最大间隔分类器的思想用数学公式表达出来。先定义几何间隔的概念，几何间隔就是在多维空间中一个多维点到一个超平面的距离，根据向量的知识可以算出来：

$$\Upsilon = \frac{\omega^T x + b}{||\omega||}$$

然后对于所有的支持向量，使他们到超平面 $\omega^T x + b$ 的距离最大，也就是

$$max_{\omega,b}\Upsilon = max_{\omega,b}\frac{\omega^T x + b}{||\omega||}$$

因为对于所有支持向量，他们 $\omega^T x + b$ 的值都是一定的，我们假设恒等于 1 ，那么上式变成了 $max_\omega \frac{1}{||\omega||} = min_\omega \frac{1}{2}||\omega||^2$,并且对于所有的样本点，满足 $y^i(\omega^T x + b) >= 1$ 的约束，因此，可以利用拉格朗日乘数法计算出它的极值。也就是求出这个超平面。

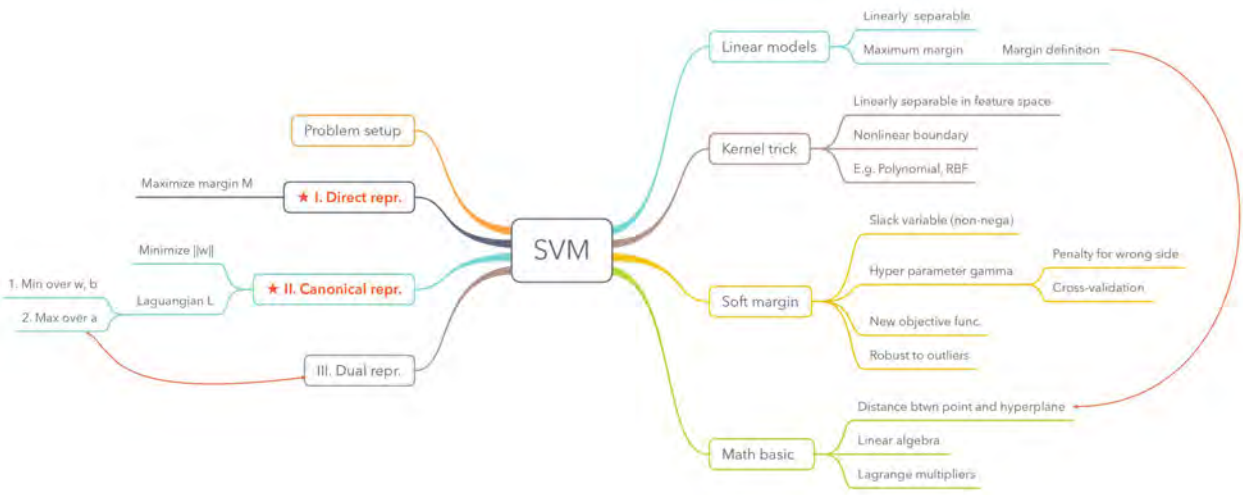
推导过程略为复杂，详细了解可以参考凸二次规划知识，结合 SMO 算法理解 SVM 计算超平面的详细过程。

总之，在计算的过程中，我们不需要了解支持向量以外的其他样本点，只需要利用相对于所有样本点来说为数不多的支持向量，就可以求出分类超平面，计算复杂度大为降低。

4、引用知乎 @靠靠靠谱 大佬的理解（这个需要的数学知识更加厉害一点）：

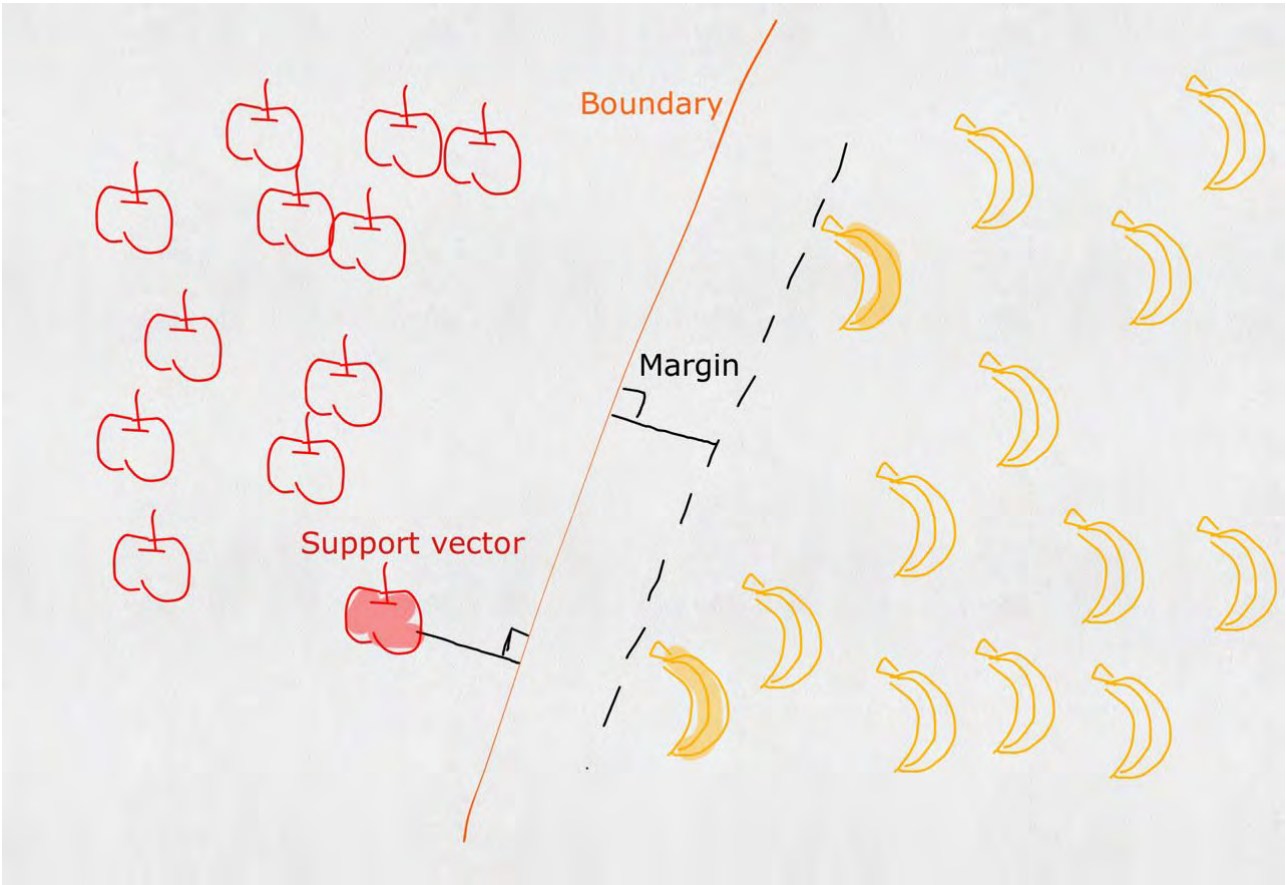
先看思维导图：

- 左边是求解基本的SVM问题
- 右边是相关扩展



什么是 SVM ？

Support Vector Machine, 一个普通的 SVM 就是一条直线罢了，用来完美划分 linearly separable 的两类。但这又不是一条普通的直线，这是无数条可以分类的直线当中最完美的，因为它恰好在两个类的中间，距离两个类的点都一样远。而所谓的 Support vector 就是这些离分界线最近的『点』。如果去掉这些点，直线多半是要改变位置的。可以说是这些 vectors （主，点点） support （谓，定义）了 machine （宾，分类器）...



所以谜底就在谜面上啊朋友们，只要找到了这些最靠近的点不就找到了 SVM 了嘛。

如果是高维的点，SVM 的分界线就是平面或者超平面。其实没有差，都是一刀切两块，我就统统叫直线了。

怎么求解 SVM ？

关于这条直线，我们知道

(1)它在离两边一样远，(2)最近距离就是到support vector，其他距离只能更远。

于是自然而然可以得到重要表达 I. direct representation

> $\arg\max_{boundary} margin(boundary)$,
> subject to 所有正确归类的苹果和香蕉到boundary的距离都$\geq margin$

（可以把 margin 看作是 boundary 的函数，并且想要找到使得是使得 margin 最大化的boundary，而 margin(*) 这个函数是：输入一个 boundary ，计算（正确分类的）所有苹果和香蕉中，到 boundary 的最小距离。）

又有最大又有最小看起来好矛盾。实际上『最大』是对这个整体使用不同 boundary 层面的最大，『最小』是在比较『点』的层面上的最小。外层在比较 boundary 找最大的 margin ，内层在比较点点找最小的距离。

其中距离，说白了就是点到直线的距离；只要定义带正负号的距离，是 {苹果+1} 面为正 {香蕉-1} 面为负的距离，互相乘上各自的 label $\in \{+1, -1\}$ ，就和谐统一民主富强了。

========== 数学表达 begin ==========
定义直线为 $y(x) = w^T x + b$
任意点x到该直线的距离为 $\frac{1}{||w||}(w^T x + b)$
对于N个训练点的信息(点的坐标，苹果还是香蕉)记为 (x_i, y_i)
上述表达也就是[1]：
$\arg\max_{w,b}\left\{\frac{1}{||w||}\min_n\left[y_i(w^T x_i + b)\right]\right\}$
不知为何这是我见过的最喜欢的写法（比心）
也可以写成[2]：
$\arg\max_{w,b,||w||=1} margin$
subject to $y_i(w^T x_i + b) \geq margin, \forall i$
$||w|| = 1$就是为了表达方便[3]，后面会取消这个限制
========== 数学表达 end ==========

到这里为止已经说完了所有关于SVM的直观了解，如果不想看求解，可以跳过下面一大段直接到 objective function 。

直接表达虽然清楚但是求解无从下手。做一些简单地等价变换（分母倒上来）可以得到 II. canonical representation （敲黑板）

> $\arg\min_{boundary} ||w||$
> subject to 所有苹果和香蕉到boundary的距离$\geq margin$

w不过是个定义直线的参数，知道这一步是等价变换出来的表达就可以了。

========== 数学表达 begin ==========
为了以后推导方便，一般写成：
$\arg\min_{w,b}\frac{1}{2}||w||^2$

subject to $y_i(w^T x_i + b) \geq 1, \forall i$
这个『1』就是一个常数，这样设置是为了以后的方便
这个选择的自由来自于直线 $y(x) = w^T x + b$ 的参数如果rescale成 kw 和 kb 不改变距离。
========== 数学表达 end ==========

要得到 III. dual representation 之前需要大概知道一下拉格朗日乘子法 (method of lagrange multiplier)，它是用在有各种约束条件(各种 "subject to")下的目标函数，也就是直接可以求导可以引出 dual representation（怎么还没完摔）

========== 数学表达 begin ==========
稍微解释一下使用拉格朗日乘子法的直观理解，不作深入讨论
$L = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n * \{y_n(w^T x_n + b) - 1\}$, 其中 $a_n > 0$ 是橙子（划去）乘子[1]
可以这样想：(1) 我们的两个任务：①对参数最小化L（解SVM要求），②对乘子又要最大化（拉格朗日乘子法要求），(2) 如果上面的约束条件成立，整个求和都是非负的，很好L是可以求最小值的；(3) 约束条件不成立，又要对乘子最大化，全身非负的L直接原地爆炸
好棒棒，所以解题一定要遵守基本法
① 先搞定第一个任务对w,b最小化L
凸优化直接取导 => 志玲（划去）置零，得到：
$w = \sum_{n=1}^{N} a_n y_n x_n$
$0 = \sum_{n=1}^{N} a_n y_n$
② 第二个任务对a最大化L，就是dual representation了
========== 数学表达 end ==========

稍微借用刚刚数学表达里面的内容看个有趣的东西：

还记得我们怎么预测一个新的水果是苹果还是香蕉吗？我们代入到分界的直线里，然后通过符号来判断。

刚刚w已经被表达出来了也就是说这个直线现在变成了：$$y(x_0) = \sum_{n=1}^{N} a_n y_n x_n^T x_0 + b$$

看似仿佛用到了所有的训练水果，但是其中 $a_n = 0$ 的水果都没有起到作用，剩下的就是小部分靠边边的 Support vectors 呀。

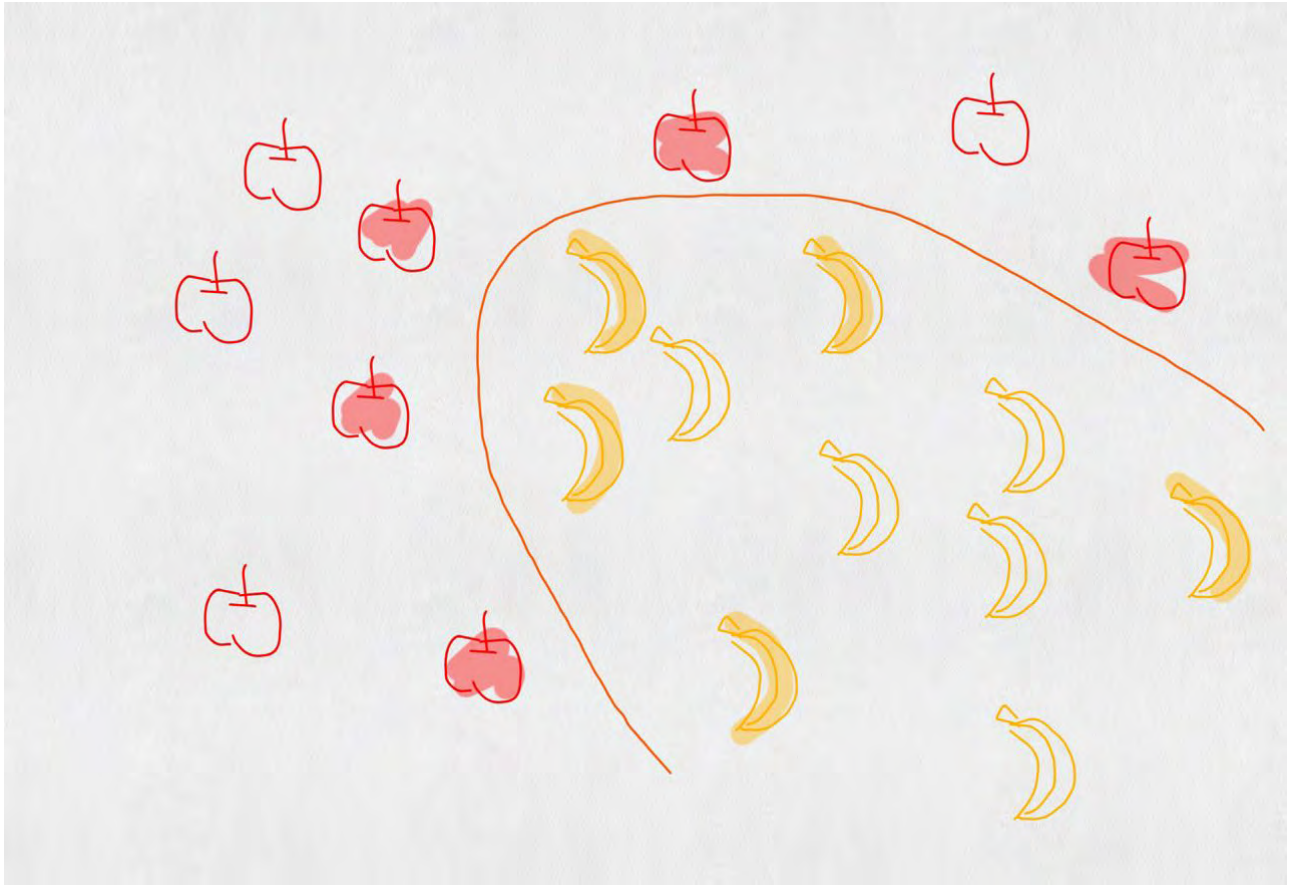
III. dual representation

把①的结果代回去就可以得到[1]：

$$\max_{all\ a_n} L(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m y_n y_m x_n^T x_m$$

$$\text{subject to } a_n \geq 0, \forall n\ , \sum_{n=1}^{N} a_n y_n = 0$$

如果香蕉和苹果不能用直线分割呢？



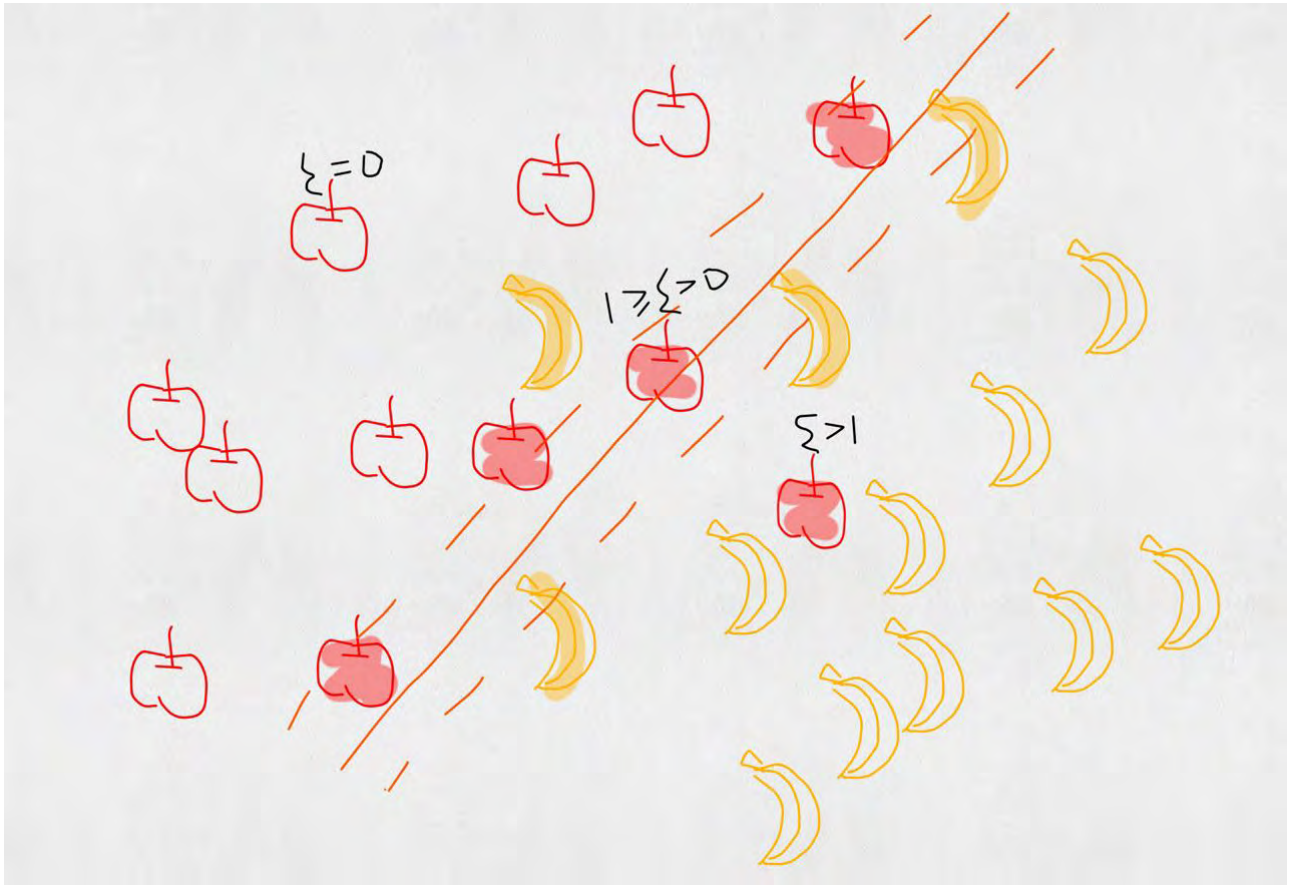
Kernel trick.

其实用直线分割的时候我们已经使用了 kernel ，那就是线性 kernel , $k(x_1, x_2) = x_1^T x_2$

如果要替换 kernel 那么把目标函数里面的内积全部替换成新的 kernel function 就好了，就是这么简单。

第一个武侠大师的比喻已经说得很直观了，低维非线性的分界线其实在高维是可以线性分割的，可以理解为——『你们是虫子！』分得开个p...（大雾）

如果香蕉和苹果有交集呢？



松弛变量 (slack variable $\xi \geq 0$)

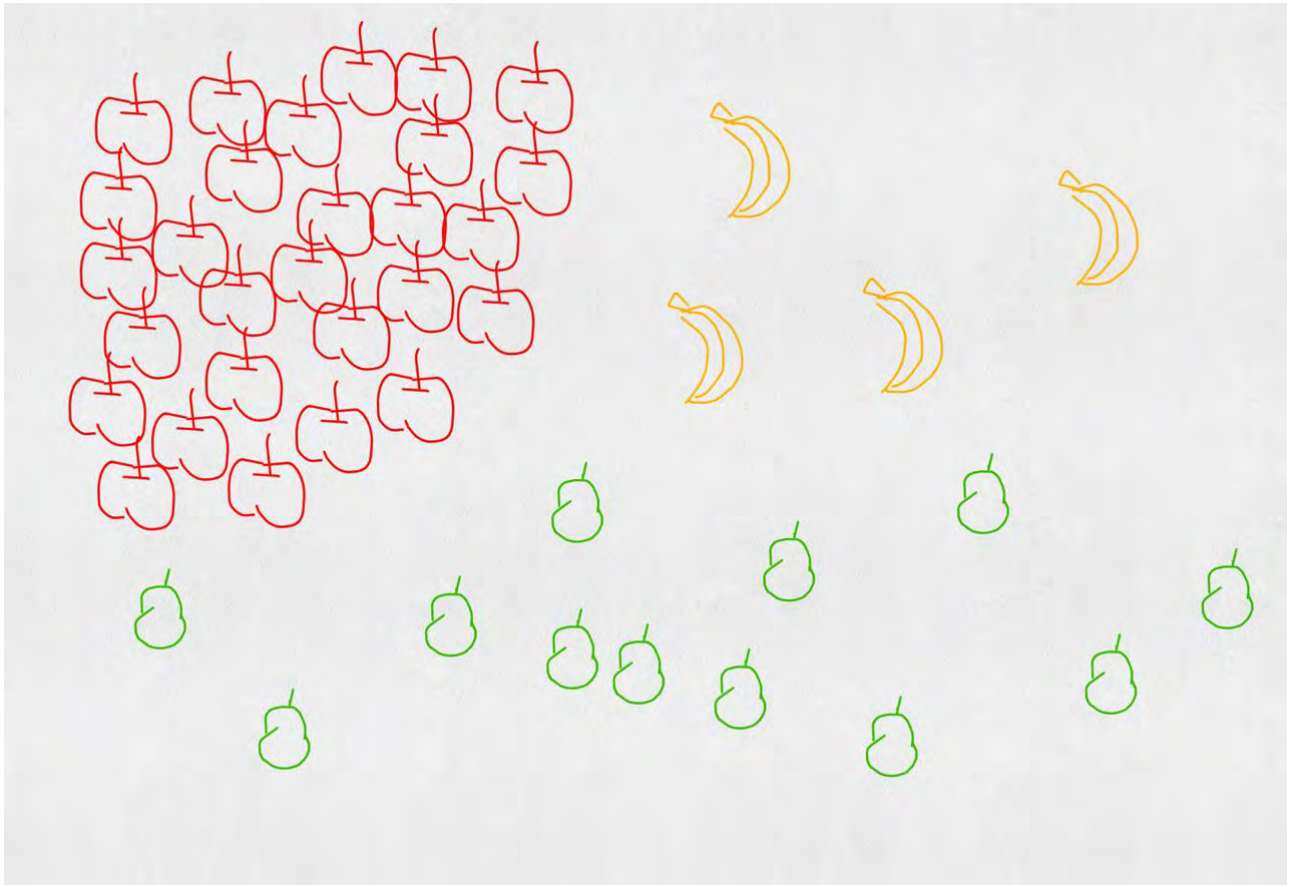
松弛变量允许错误的分类，但是要付出代价。图中以苹果为例，错误分类的苹果$\xi > 1$；在margin当中但是正确分类的苹果$0 < \xi \leq 1$；正确分类并且在margin外面的苹果$\xi = 0$。可以看出每一个数据都有一一对应的惩罚。

对于这一次整体的惩罚力度，要另外使用一个超参数 (γ) 来衡量这一次分类的penalty程度。

从新的目标函数里可见一斑[1]： $\min \frac{1}{2}||w||^2 + \frac{\gamma}{2}\sum_{n=1}^{N}\xi_n^2$

（约束条件略）

如果还有梨呢？



可以每个类别做一次 SVM：是苹果还是不是苹果？是香蕉还是不是香蕉？是梨子还是不是梨子？从中选出可能性最大的。这是 one-versus-the-rest approach。

也可以两两做一次 SVM：是苹果还是香蕉？是香蕉还是梨子？是梨子还是苹果？最后三个分类器投票决定。这是 one-versus-one approace。

但这其实都多多少少有问题，比如苹果特别多，香蕉特别少，我就无脑判断为苹果也不会错太多；多个分类器要放到一个台面上，万一他们的 scale 没有在一个台面上也未可知。

课后题： 1、vector 不愿意 support 怎么办？ 2、苹果好吃还是香蕉好吃？

最后送一张图我好爱哈哈哈 (Credit: Burr Settles)

[1] Bishop C M. Pattern recognition[J]. Machine Learning, 2006, 128.

[2] Friedman J, Hastie T, Tibshirani R. The elements of statistical learning[M]. Springer, Berlin: Springer series in statistics, 2001.

[3] James G, Witten D, Hastie T, et al. An introduction to statistical learning[M]. New York: springer, 2013.

理解和应用

1、DataMining （数据挖掘）

做数据挖掘应用的一种重要算法，也是效果最好的分类算法之一。

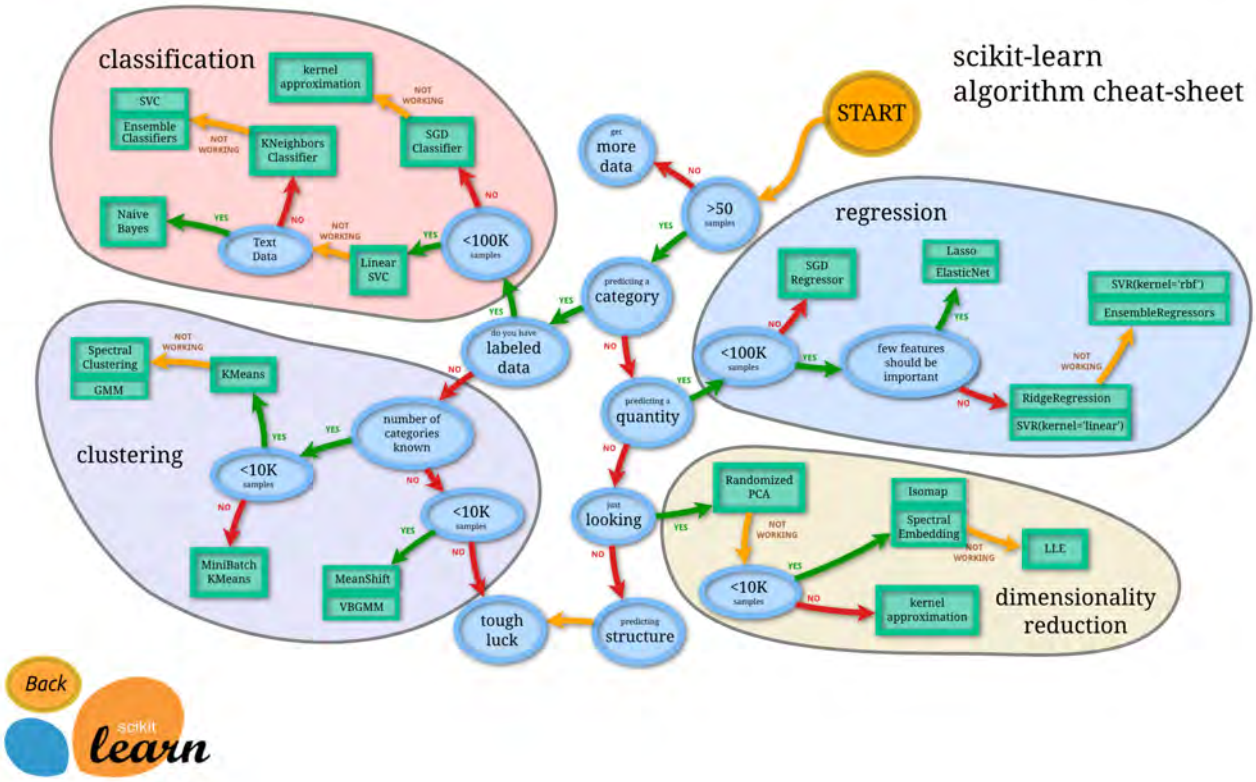
举个例子，就是尽量把样本中的从更高纬度看起来在一起的样本合在一起，比如在一维（直线）空间里的样本从二维平面上可以分成不同类别，而在二维平面上分散的样本如果从第三维空间上来看就可以对他们做分类。

支持向量机算法目的是找出最优超平面，使分类间隔最大，要求不但正确分开，而且使分类间隔最大，在两类样本中离分类平面最近且位于平行于最优超平面的超平面上的点就是支持向量，为找到最优超平面只要找到所有支持向量即可。

对于非线性支持向量机，通常做法是把线性不可分转化成线性可分，通过一个非线性映射将低维输入空间中的数据特性映射到高维线性特征空间中，在高维空间中求线性最优分类超平面。

2、scikit-learn (sklearn)

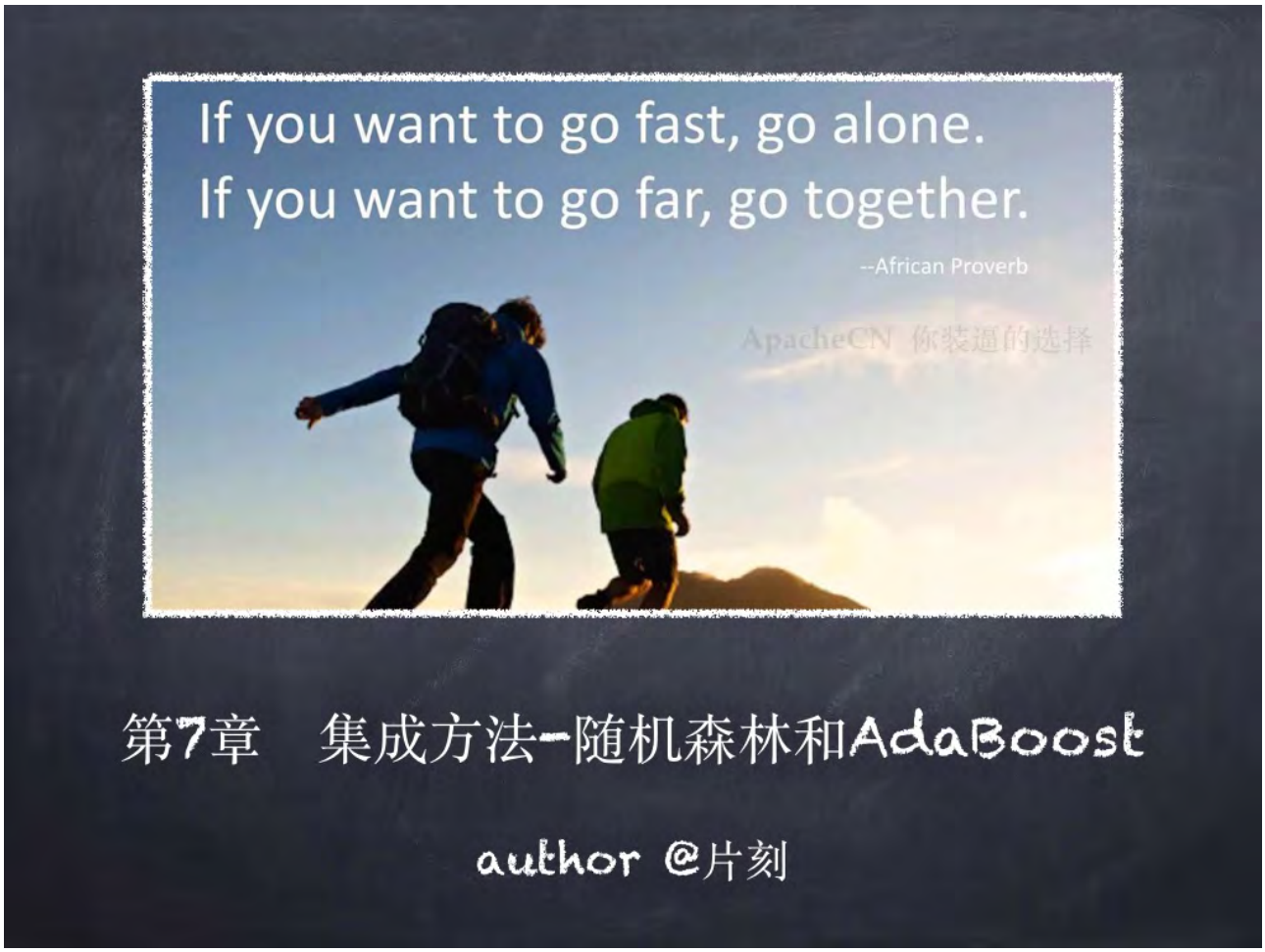
SVM 的基本原理基本上已经说的差不多了，下面咱们就来看看 SVM 在实际应用该如何使用了。幸运的是，在 python 下面，sklearn 提供了一个非常好用的机器学习算法，我们调用相关的包就好啦。



小结

学习 SVM 需要有耐心，当初研究这个部分的时候，炼哥（github jiangzhonglian），法超大佬（github geekidentity），羊三大佬（github sheepmen），庭哥（github wangyangting）都花费了好长时间，我只能躲在角落发抖....

第7章 集成方法 ensemble method



集成方法: ensemble method （元算法: meta algorithm ）概述

- 概念：是对其他算法进行组合的一种形式。
- 通俗来说： 当做重要决定时，大家可能都会考虑吸取多个专家而不只是一个人的意见。 机器学习处理问题时又何尝不是如此？ 这就是集成方法背后的思想。
- 集成方法：
    1. 投票选举(bagging: 自举汇聚法 bootstrap aggregating): 是基于数据随机重抽样分类器构造的方法
    2. 再学习(boosting): 是基于所有分类器的加权求和的方法

集成方法 场景

目前 bagging 方法最流行的版本是: 随机森林(random forest)
选男友：美女选择择偶对象的时候，会问几个闺蜜的建议，最后选择一个综合得分最高的一个作为男朋友

目前 boosting 方法最流行的版本是: AdaBoost
追女友：3个帅哥追同一个美女，第1个帅哥失败->(传授经验：姓名、家庭情况) 第2个帅哥失败->(传授经验：兴趣爱好、性格特点) 第3个帅哥成功

> bagging 和 boosting 区别是什么？

1. bagging 是一种与 boosting 很类似的技术, 所使用的多个分类器的类型（数据量和特征量）都是一致的。
2. bagging 是由不同的分类器（1.数据随机化 2.特征随机化）经过训练，综合得出的出现最多分类结果；boosting 是通过调整已有分类器错分的那些数据来获得新的分类器，得出目前最优的结果。
3. bagging 中的分类器权重是相等的；而 boosting 中的分类器加权求和，所以权重并不相等，每个权重代表的是其对应分类器在上一轮迭代中的成功度。

随机森林

随机森林 概述

- 随机森林指的是利用多棵树对样本进行训练并预测的一种分类器。
- 决策树相当于一个大师，通过自己在数据集中学到的知识用于新数据的分类。但是俗话说得好，一个诸葛亮，玩不过三个臭皮匠。随机森林就是希望构建多个臭皮匠，希望最终的分类效果能够超过单个大师的一种算法。

随机森林 原理

那随机森林具体如何构建呢？
有两个方面：

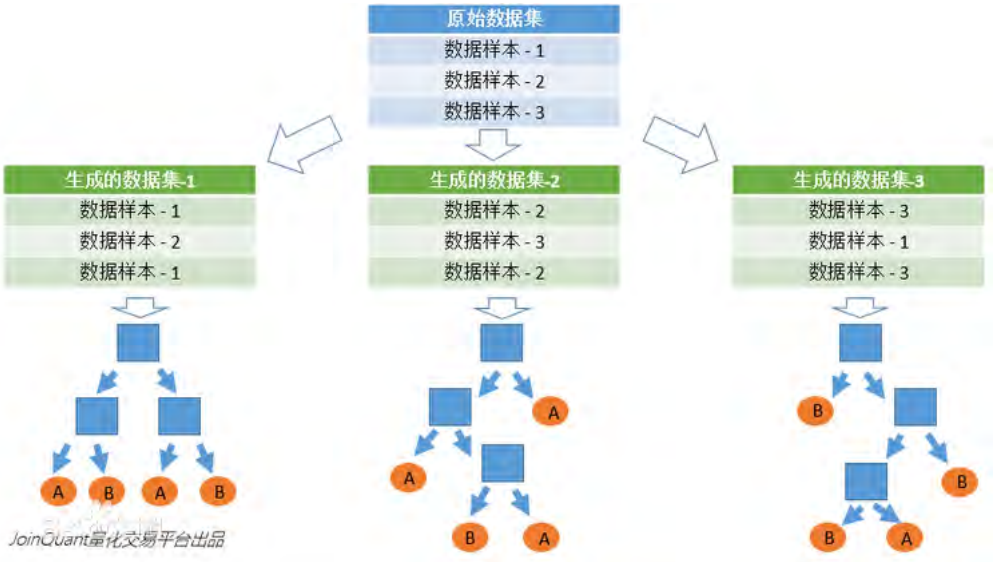
1. 数据的随机性化
2. 待选特征的随机化

使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

> 数据的随机化：使得随机森林中的决策树更普遍化一点，适合更多的场景。

（有放回的准确率在：70% 以上， 无放回的准确率在：60% 以上）

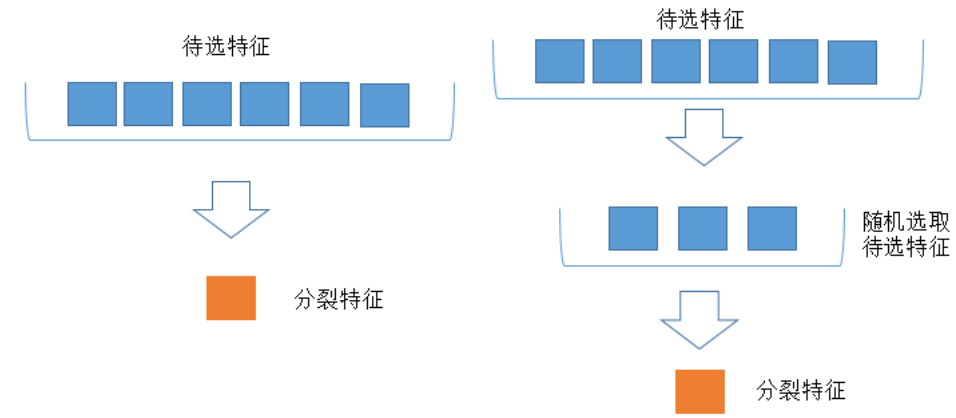
1. 采取有放回的抽样方式 构造子数据集，保证不同子集之间的数量级一样（不同子集 / 同一子集 之间的元素可以重复）
2. 利用子数据集来构建子决策树，将这个数据放到每个子决策树中，每个子决策树输出一个结果。
3. 然后统计子决策树的投票结果，得到最终的分类 就是 随机森林的输出结果。
4. 如下图，假设随机森林中有3棵子决策树，2棵子树的分类结果是A类，1棵子树的分类结果是B类，那么随机森林的分类结果就是A类。



> 待选特征的随机化

1. 子树从所有的待选特征中随机选取一定的特征。
2. 在选取的特征中选取最优的特征。

下图中，蓝色的方块代表所有可以被选择的特征，也就是目前的待选特征；黄色的方块是分裂特征。
左边是一棵决策树的特征选取过程，通过在待选特征中选取最优的分裂特征（别忘了前文提到的ID3算法，C4.5算法，CART算法等等），完成分裂。
右边是一个随机森林中的子树的特征选取过程。



决策树选取分裂特征过程　　随机森林子树选取分裂特征过程

随机森林 开发流程

```
收集数据：任何方法
准备数据：转换样本集
分析数据：任何方法
训练算法：通过数据随机化和特征随机化，进行多实例的分类评估
测试算法：计算错误率
使用算法：输入样本数据，然后运行 随机森林 算法判断输入数据分类属于哪个分类，最后对计算出的分类执行后续处理
```

随机森林 算法特点

```
优点：几乎不需要输入准备、可实现隐式特征选择、训练速度非常快、其他模型很难超越、很难建立一个糟糕的随机森林模型、大量优秀、免费以及开源的实现。
缺点：劣势在于模型大小、是个很难去解释的黑盒子。
适用数据范围：数值型和标称型
```

项目案例: 声纳信号分类

项目概述

这是 Gorman 和 Sejnowski 在研究使用神经网络的声纳信号分类中使用的数据集。任务是训练一个模型来区分声纳信号。

开发流程

```
收集数据：提供的文本文件
准备数据：转换样本集
分析数据：手工检查数据
训练算法：在数据上，利用 random_forest() 函数进行优化评估，返回模型的综合分类结果
测试算法：在采用自定义 n_folds 份随机重抽样 进行测试评估，得出综合的预测评分
使用算法：若你感兴趣可以构建完整的应用程序，从案例进行封装，也可以参考我们的代码
```

收集数据：提供的文本文件

样本数据：sonar-all-data.txt

```
0.02,0.0371,0.0428,0.0207,0.0954,0.0986,0.1539,0.1601,0.3109,0.2111,0.1609,0.1582,0.2238,0.0645,0.066,0.2273,0.31,0.2999,0.5078,0.4797,0.5783,0.
0.0453,0.0523,0.0843,0.0689,0.1183,0.2583,0.2156,0.3481,0.3337,0.2872,0.4918,0.6552,0.6919,0.7797,0.7464,0.9444,1,0.8874,0.8024,0.7818,0.5212,0.
0.0262,0.0582,0.1099,0.1083,0.0974,0.228,0.2431,0.3771,0.5598,0.6194,0.6333,0.706,0.5544,0.532,0.6479,0.6931,0.6759,0.7551,0.8929,0.8619,0.7974,
```

准备数据：转换样本集

```
# 导入csv文件
def loadDataSet(filename):
    dataset = []
    with open(filename, 'r') as fr:
        for line in fr.readlines():
            if not line:
                continue
            lineArr = []
            for featrue in line.split(','):
                # strip()返回移除字符串头尾指定的字符生成的新字符串
                str_f = featrue.strip()
                if str_f.isdigit(): # 判断是否是数字
```

```
                    # 将数据集的第column列转换成float形式
                    lineArr.append(float(str_f))
                else:
                    # 添加分类标签
                    lineArr.append(str_f)
            dataset.append(lineArr)
    return dataset
```

> 分析数据：手工检查数据

> 训练算法：在数据上，利用 random_forest() 函数进行优化评估，返回模型的综合分类结果

- 样本数据随机无放回抽样-用于交叉验证

```
def cross_validation_split(dataset, n_folds):
    """cross_validation_split(将数据集进行抽重抽样 n_folds 份，数据可以重复重复抽取)

    Args:
        dataset     原始数据集
        n_folds     数据集dataset分成n_flods份
    Returns:
        dataset_split    list集合，存放的是：将数据集进行抽重抽样 n_folds 份，数据可以重复重复抽取
    """
    dataset_split = list()
    dataset_copy = list(dataset)       # 复制一份 dataset,防止 dataset 的内容改变
    fold_size = len(dataset) / n_folds
    for i in range(n_folds):
        fold = list()                  # 每次循环 fold 清零，防止重复导入 dataset_split
        while len(fold) < fold_size:   # 这里不能用 if，if 只是在第一次判断时起作用，while 执行循环，直到条件不成立
            # 有放回的随机采样，有一些样本被重复采样，从而在训练集中多次出现，有的则从未在训练集中出现，此则自助采样法。从而保证每棵决策树训练集的
            index = randrange(len(dataset_copy))
            # 将对应索引 index 的内容从 dataset_copy 中导出，并将该内容从 dataset_copy 中删除。
            # pop() 函数用于移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。
            fold.append(dataset_copy.pop(index))  # 无放回的方式
            # fold.append(dataset_copy[index])  # 有放回的方式
        dataset_split.append(fold)
    # 由dataset分割出的n_folds个数据构成的列表，为了用于交叉验证
    return dataset_split
```

- 训练数据集随机化

```
# Create a random subsample from the dataset with replacement
def subsample(dataset, ratio):   # 创建数据集的随机子样本
    """random_forest(评估算法性能，返回模型得分)

    Args:
        dataset         训练数据集
        ratio           训练数据集的样本比例
    Returns:
        sample          随机抽样的训练样本
    """

    sample = list()
    # 训练样本的按比例抽样。
    # round() 方法返回浮点数x的四舍五入值。
    n_sample = round(len(dataset) * ratio)
    while len(sample) < n_sample:
        # 有放回的随机采样，有一些样本被重复采样，从而在训练集中多次出现，有的则从未在训练集中出现，此则自助采样法。从而保证每棵决策树训练集的差异
        index = randrange(len(dataset))
        sample.append(dataset[index])
    return sample
```

- 特征随机化

```
# 找出分割数据集的最优特征，得到最优的特征 index，特征值 row[index]，以及分割完的数据 groups（left, right）
def get_split(dataset, n_features):
    class_values = list(set(row[-1] for row in dataset))  # class_values =[0, 1]
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    features = list()
    while len(features) < n_features:
        index = randrange(len(dataset[0])-1)  # 往 features 添加 n_features 个特征（ n_feature 等于特征数的根号），特征索引从 dataset 中随
        if index not in features:
            features.append(index)
    for index in features:                    # 在 n_features 个特征中选出最优的特征索引，并没有遍历所有特征，从而保证了每课决策树的差异性
        for row in dataset:
            groups = test_split(index, row[index], dataset)  # groups=(left, right), row[index] 遍历每一行 index 索引下的特征值作为分类值
            gini = gini_index(groups, class_values)
            # 左右两边的数量越一样，说明数据区分度不高，gini系数越大
            if gini < b_score:
```

```
                b_index, b_value, b_score, b_groups = index, row[index], gini, groups  # 最后得到最优的分类特征 b_index,分类特征值 b_val
    # print b_score
    return {'index': b_index, 'value': b_value, 'groups': b_groups}
```

- 随机森林

```
# Random Forest Algorithm
def random_forest(train, test, max_depth, min_size, sample_size, n_trees, n_features):
    """random_forest(评估算法性能，返回模型得分)

    Args:
        train           训练数据集
        test            测试数据集
        max_depth       决策树深度不能太深，不然容易导致过拟合
        min_size        叶子节点的大小
        sample_size     训练数据集的样本比例
        n_trees         决策树的个数
        n_features      选取的特征的个数
    Returns:
        predictions     每一行的预测结果，bagging 预测最后的分类结果
    """

    trees = list()
    # n_trees 表示决策树的数量
    for i in range(n_trees):
        # 随机抽样的训练样本， 随机采样保证了每棵决策树训练集的差异性
        sample = subsample(train, sample_size)
        # 创建一个决策树
        tree = build_tree(sample, max_depth, min_size, n_features)
        trees.append(tree)

    # 每一行的预测结果，bagging 预测最后的分类结果
    predictions = [bagging_predict(trees, row) for row in test]
    return predictions
```

> 测试算法：在采用自定义 n_folds 份随机重抽样 进行测试评估，得出综合的预测评分。

- 计算随机森林的预测结果的正确率

```
# 评估算法性能，返回模型得分
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    """evaluate_algorithm(评估算法性能，返回模型得分)

    Args:
        dataset     原始数据集
        algorithm   使用的算法
        n_folds     数据的份数
        *args       其他的参数
    Returns:
        scores      模型得分
    """

    # 将数据集进行随机抽样，分成 n_folds 份，数据无重复的抽取
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    # 每次循环从 folds 从取出一个 fold 作为测试集，其余作为训练集，遍历整个 folds ，实现交叉验证
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        # 将多个 fold 列表组合成一个 train_set 列表, 类似 union all
        """
        In [20]: l1=[[1, 2, 'a'], [11, 22, 'b']]
        In [21]: l2=[[3, 4, 'c'], [33, 44, 'd']]
        In [22]: l=[]
        In [23]: l.append(l1)
        In [24]: l.append(l2)
        In [25]: l
        Out[25]: [[[1, 2, 'a'], [11, 22, 'b']], [[3, 4, 'c'], [33, 44, 'd']]]
        In [26]: sum(l, [])
        Out[26]: [[1, 2, 'a'], [11, 22, 'b'], [3, 4, 'c'], [33, 44, 'd']]
        """
        train_set = sum(train_set, [])
        test_set = list()
        # fold 表示从原始数据集 dataset 提取出来的测试集
        for row in fold:
            row_copy = list(row)
            row_copy[-1] = None
            test_set.append(row_copy)
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
```

```
        # 计算随机森林的预测结果的正确率
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores
```

> 使用算法：若你感兴趣可以构建完整的应用程序，从案例进行封装，也可以参考我们的代码

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/7.RandomForest/randomForest.py

AdaBoost

AdaBoost (adaptive boosting:　自适应 boosting) 概述

能否使用弱分类器和多个实例来构建一个强分类器？ 这是一个非常有趣的理论问题。

AdaBoost 原理

> AdaBoost 工作原理

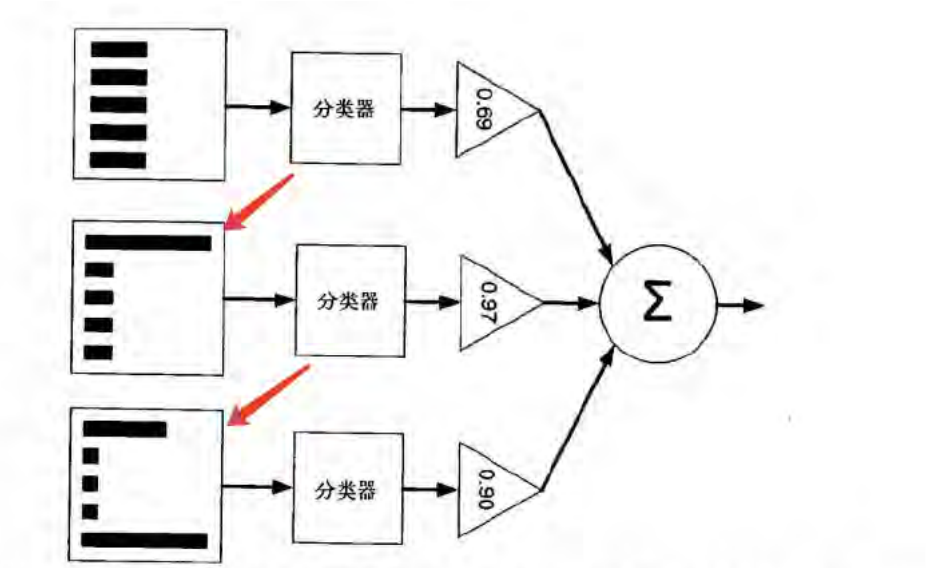


图7-1 AdaBoost算法的示意图。左边是数据集，其中直方图的不同宽度表示每个样例上的不同权重。在经过一个分类器之后，加权的预测结果会通过三角形中的alpha值进行加权。每个三角形中输出的加权结果在圆形中求和，从而得到最终的输出结果

> AdaBoost 开发流程

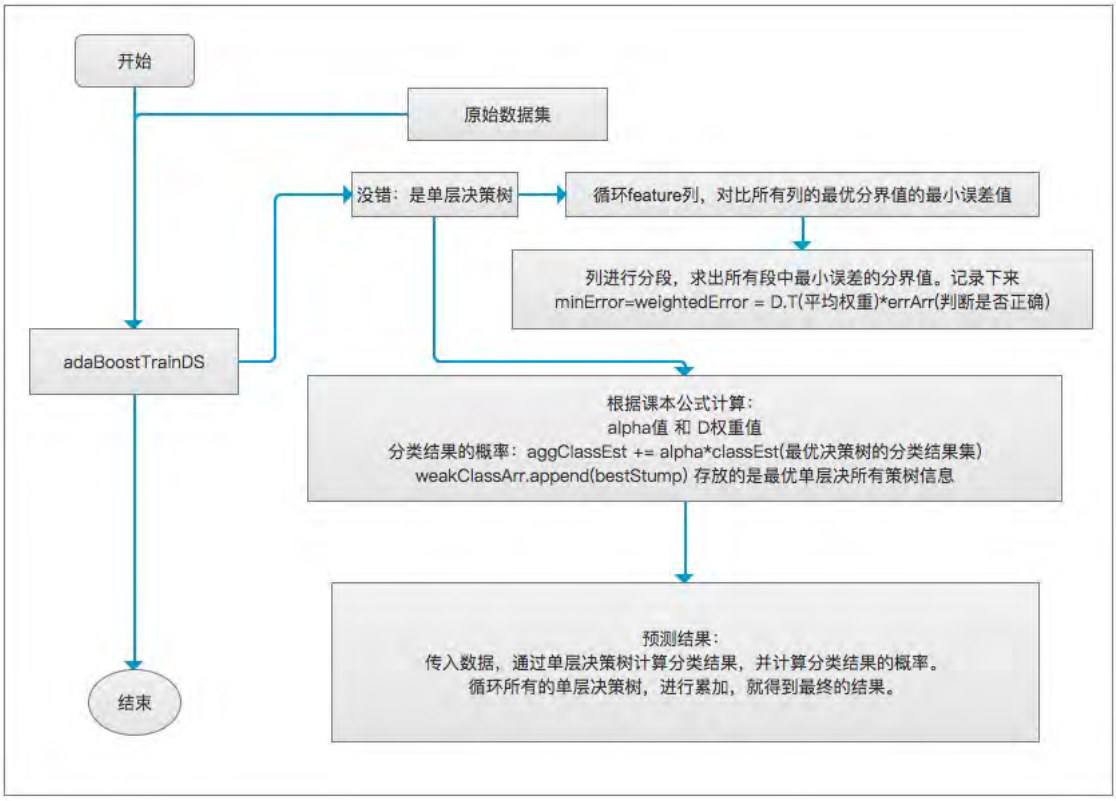
```
收集数据：可以使用任意方法
准备数据：依赖于所使用的弱分类器类型，本章使用的是单层决策树，这种分类器可以处理任何数据类型。
    当然也可以使用任意分类器作为弱分类器，第2章到第6章中的任一分类器都可以充当弱分类器。
    作为弱分类器，简单分类器的效果更好。
分析数据：可以使用任意方法。
训练算法：AdaBoost 的大部分时间都用在训练上，分类器将多次在同一数据集上训练弱分类器。
测试算法：计算分类的错误率。
使用算法：通SVM一样，AdaBoost 预测两个类别中的一个。如果想把它应用到多个类别的场景，那么就要像多类 SVM 中的做法一样对 AdaBoost 进行修改。
```

> AdaBoost 算法特点

```
* 优点：泛化（由具体的、个别的扩大为一般的）错误率低，易编码，可以应用在大部分分类器上，无参数调节。
* 缺点：对离群点敏感。
* 适用数据类型：数值型和标称型数据。
```

项目案例: 马疝病的预测

> 项目流程图



基于单层决策树构建弱分类器

- 单层决策树(decision stump, 也称决策树桩)是一种简单的决策树。

项目概述

预测患有疝气病的马的存活问题，这里的数据包括368个样本和28个特征，疝气病是描述马胃肠痛的术语，然而，这种病并不一定源自马的胃肠问题，其他问题也可能引发疝气病，该数据集中包含了医院检测马疝气病的一些指标，有的指标比较主观，有的指标难以测量，例如马的疼痛级别。另外，除了部分指标主观和难以测量之外，该数据还存在一个问题，数据集中有30%的值是缺失的。

开发流程

```
收集数据：提供的文本文件
准备数据：确保类别标签是+1和-1，而非1和0
分析数据：统计分析
训练算法：在数据上，利用 adaBoostTrainDS() 函数训练出一系列的分类器
测试算法：我们拥有两个数据集。在不采用随机抽样的方法下，我们就会对 AdaBoost 和 Logistic 回归的结果进行完全对等的比较
使用算法：观察该例子上的错误率。不过，也可以构建一个 Web 网站，让驯马师输入马的症状然后预测马是否会死去
```

> 收集数据：提供的文本文件

训练数据：horseColicTraining.txt
测试数据：horseColicTest.txt

```
2.000000	1.000000	38.500000	66.000000	28.000000	3.000000	3.000000	0.000000	2.000000
1.000000	1.000000	39.200000	88.000000	20.000000	0.000000	0.000000	4.000000	1.000000
2.000000	1.000000	38.300000	40.000000	24.000000	1.000000	1.000000	3.000000	1.000000
```

> 准备数据：确保类别标签是+1和-1，而非1和0

```
def loadDataSet(fileName):
    # 获取 feature 的数量，便于获取
    numFeat = len(open(fileName).readline().split('\t'))
    dataArr = []
    labelArr = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = []
        curLine = line.strip().split('\t')
        for i in range(numFeat-1):
            lineArr.append(float(curLine[i]))
        dataArr.append(lineArr)
        labelArr.append(float(curLine[-1]))
    return dataArr, labelArr
```

> 分析数据：统计分析

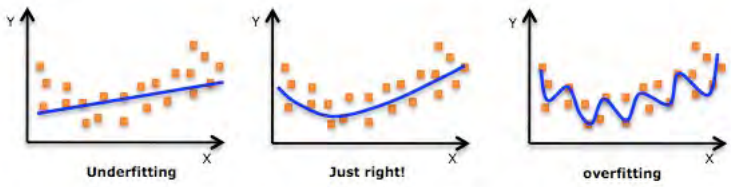
过拟合(overfitting, 也称为过学习)

- 发现测试错误率在达到一个最小值之后有开始上升，这种现象称为过拟合。

表7-1 不同弱分类器数目情况下的AdaBoost测试和分类错误率。该数据集是个难数据集。通常情况下，AdaBoost会达到一个稳定的测试错误率，而并不会随分类器数目的增多而提高

分类器数目	训练错误率（%）	测试错误率（%）
1	0.28	0.27
10	0.23	0.24
50	0.19	0.21
100	0.19	0.22
500	0.16	0.25
1000	0.14	0.31
10000	0.11	0.33

- 通俗来说：就是把一些噪音数据也拟合进去的，如下图。



> 训练算法：在数据上，利用 adaBoostTrainDS() 函数训练出一系列的分类器

```
def adaBoostTrainDS(dataArr, labelArr, numIt=40):
    """adaBoostTrainDS(adaBoost训练过程放大)
    Args:
        dataArr   特征标签集合
        labelArr  分类标签集合
        numIt     实例数
    Returns:
        weakClassArr  弱分类器的集合
        aggClassEst   预测的分类结果值
    """
    weakClassArr = []
    m = shape(dataArr)[0]
    # 初始化 D，设置每个样本的权重值，平均分为m份
    D = mat(ones((m, 1))/m)
    aggClassEst = mat(zeros((m, 1)))
    for i in range(numIt):
        # 得到决策树的模型
        bestStump, error, classEst = buildStump(dataArr, labelArr, D)

        # alpha目的主要是计算每一个分类器实例的权重(组合就是分类结果)
        # 计算每个分类器的alpha权重值
        alpha = float(0.5*log((1.0-error)/max(error, 1e-16)))
        bestStump['alpha'] = alpha
        # store Stump Params in Array
        weakClassArr.append(bestStump)

        print "alpha=%s, classEst=%s, bestStump=%s, error=%s " % (alpha, classEst.T, bestStump, error)
        # 分类正确：乘积为1，不会影响结果，-1主要是下面求e的-alpha次方
        # 分类错误：乘积为 -1，结果会受影响，所以也乘以 -1
        expon = multiply(-1*alpha*mat(labelArr).T, classEst)
        print '(-1取反)预测值expon=', expon.T
        # 计算e的expon次方，然后计算得到一个综合的概率的值
        # 结果发现： 判断错误的样本，D对于的样本权重值会变大。
        D = multiply(D, exp(expon))
        D = D/D.sum()

        # 预测的分类结果值，在上一轮结果的基础上，进行加和操作
        print '当前的分类结果：', alpha*classEst.T
        aggClassEst += alpha*classEst
        print "叠加后的分类结果aggClassEst: ", aggClassEst.T
        # sign 判断正为1， 0为0， 负为-1，通过最终加和的权重值，判断符号。
        # 结果为：错误的样本标签集合，因为是 !=,那么结果就是0 正，1 负
        aggErrors = multiply(sign(aggClassEst) != mat(labelArr).T, ones((m, 1)))
        errorRate = aggErrors.sum()/m
        # print "total error=%s " % (errorRate)
        if errorRate == 0.0:
```

```
            break
    return weakClassArr, aggClassEst
```

```
发现：
alpha （模型权重）目的主要是计算每一个分类器实例的权重(加和就是分类结果)
  分类的权重值：最大的值= alpha 的加和，最小值=-最大值
D （样本权重）的目的是为了计算错误概率： weightedError = D.T*errArr，求最佳分类器
  样本的权重值：如果一个值误判的几率越小，那么 D 的样本权重越小
```

AdaBoost是adaptive boosting（自适应boosting）的缩写，其运行过程如下：训练数据中的每个样本，并赋予其一个权重，这些权重构成了向量D。一开始，这些权重都初始化成相等值。首先在训练数据上训练出一个弱分类器并计算该分类器的错误率，然后在同一数据集上再次训练弱分类器。在分类器的第二次训练当中，将会重新调整每个样本的权重，其中第一次分对的样本的权重将会降低，而第一次分错的样本的权重将会提高。为了从所有弱分类器中得到最终的分类结果，AdaBoost为每个分类器都分配了一个权重值alpha，这些alpha值是基于每个弱分类器的错误率进行计算的。其中，错误率ε的定义为：

$$\varepsilon = \frac{\text{未正确分类的样本数目}}{\text{所有样本数目}}$$

而alpha的计算公式如下：

$$\alpha = \frac{1}{2}\ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

计算出alpha值之后，可以对权重向量D进行更新，以使得那些正确分类的样本的权重降低而错分样本的权重升高。D的计算方法如下。

如果某个样本被正确分类，那么该样本的权重更改为：

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{\text{Sum}(D)}$$

而如果某个样本被错分，那么该样本的权重更改为：

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{\text{Sum}(D)}$$

在计算出D之后，AdaBoost又开始进入下一轮迭代。AdaBoost算法会不断地重复训练和调整权重的过程，直到训练错误率为0或者弱分类器的数目达到用户的指定值为止。

> 测试算法：我们拥有两个数据集。在不采用随机抽样的方法下，我们就会对 AdaBoost 和 Logistic 回归的结果进行完全对等的比较。

```
def adaClassify(datToClass, classifierArr):
    """adaClassify(ada分类测试)
    Args:
        datToClass     多个待分类的样例
        classifierArr  弱分类器的集合
    Returns:
        sign(aggClassEst) 分类结果
    """
    # do stuff similar to last aggClassEst in adaBoostTrainDS
    dataMat = mat(datToClass)
    m = shape(dataMat)[0]
    aggClassEst = mat(zeros((m, 1)))

    # 循环 多个分类器
    for i in range(len(classifierArr)):
        # 前提： 我们已经知道了最佳的分类器的实例
        # 通过分类器来核算每一次的分类结果，然后通过alpha*每一次的结果 得到最后的权重加和的值。
        classEst = stumpClassify(dataMat, classifierArr[i]['dim'], classifierArr[i]['thresh'], classifierArr[i]['ineq'])
        aggClassEst += classifierArr[i]['alpha']*classEst
    return sign(aggClassEst)
```

> 使用算法：观察该例子上的错误率。不过，也可以构建一个 Web 网站，让驯马师输入马的症状然后预测马是否会死去。

```
# 马疝病数据集
# 训练集合
dataArr, labelArr = loadDataSet("input/7.AdaBoost/horseColicTraining2.txt")
weakClassArr, aggClassEst = adaBoostTrainDS(dataArr, labelArr, 40)
print weakClassArr, '\n-----\n', aggClassEst.T
# 计算ROC下面的AUC的面积大小
plotROC(aggClassEst.T, labelArr)
# 测试集合
dataArrTest, labelArrTest = loadDataSet("input/7.AdaBoost/horseColicTest2.txt")
m = shape(dataArrTest)[0]
predicting10 = adaClassify(dataArrTest, weakClassArr)
errArr = mat(ones((m, 1)))
```

```
# 测试：计算总样本数，错误样本数，错误率
print m, errArr[predicting10 != mat(labelArrTest).T].sum(), errArr[predicting10 != mat(labelArrTest).T].sum()/m
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/7.AdaBoost/adaboost.py

要点补充

> 非均衡现象：

`在分类器训练时，正例数目和反例数目不相等（相差很大）`

- 判断马是否能继续生存(不可误杀)
- 过滤垃圾邮件(不可漏判)
- 不能放过传染病的人
- 不能随便认为别人犯罪

> ROC 评估方法

- ROC 曲线: 最佳的分类器应该尽可能地处于左上角

图7-3给出了一条ROC曲线的例子。

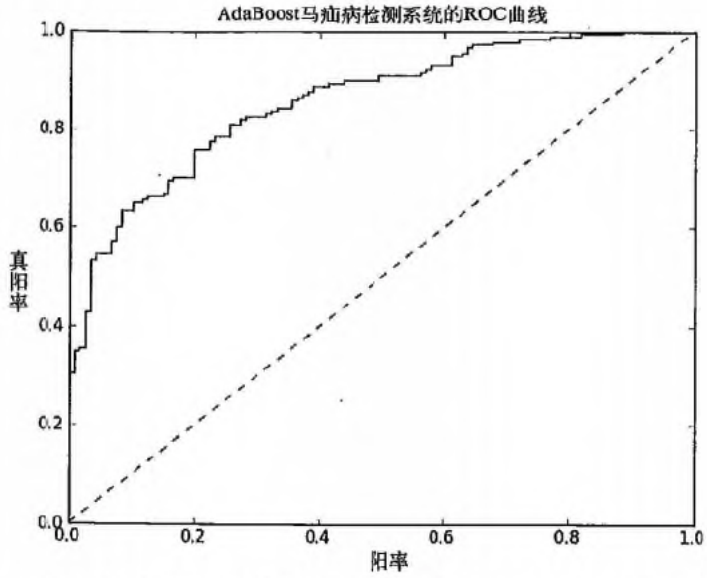


图7-3 利用10个单层决策树的AdaBoost马疝病检测系统的ROC曲线

在图7-3的ROC曲线中，给出了两条线，一条虚线一条实线。图中的横轴是伪正例的比例（假阳率=FP/(FP+TN)），而纵轴是真正例的比例（真阳率=TP/(TP+FN)）。ROC曲线给出的是当阈值变化时假阳率和真阳率的变化情况。左下角的点所对应的是将所有样例判为反例的情况，而右上角的点对应的则是将所有样例判为正例的情况。虚线给出的是随机猜测的结果曲线。

- 对不同的 ROC 曲线进行比较的一个指标是曲线下的面积(Area Unser the Curve, AUC).
- AUC 给出的是分类器的平均性能值，当然它并不能完全代替对整条曲线的观察。
- 一个完美分类器的 AUC 为1，而随机猜测的 AUC 则为0.5。

> 代价函数

- 基于代价函数的分类器决策控制：`TP*(-5)+FN*1+FP*50+TN*0`

表7-3 一个二类问题的混淆矩阵，其中的输出采用了不同的类别标签

		预测结果 +1	预测结果 −1
真实结果	+1	真正例（TP）	伪反例（FN）
	−1	伪正例（FP）	真反例（TN）

		预测结果 +1	预测结果 −1
真实结果	+1	−5	1
	−1	50	0

> 抽样

- 欠抽样(undersampling)或者过抽样(oversampling)
  - 欠抽样: 意味着删除样例
  - 过抽样: 意味着复制样例(重复使用)

- 作者：片刻
- GitHub地址: https://github.com/apachecn/MachineLearning

第8章 预测数值型数据：回归

回归（Regression ）概述

我们前边提到的分类的目标变量是标称型数据，而回归则是对连续型的数据做出处理，回归的目的是预测数值型数据的目标值。

回归 场景

回归的目的是预测数值型的目标值。最直接的办法是依据输入写出一个目标值的计算公式。

假如你想要预测兰博基尼跑车的功率大小，可能会这样计算:

HorsePower = 0.0015 * annualSalary - 0.99 * hoursListeningToPublicRadio

这就是所谓的 回归方程(regression equation)，其中的 0.0015 和 -0.99 称作 回归系数（regression weights），求这些回归系数的过程就是回归。一旦有了这些回归系数，再给定输入，做预测就非常容易了。具体的做法是用回归系数乘以输入值，再将结果全部加在一起，就得到了预测值。我们这里所说的，回归系数是一个向量，输入也是向量，这些运算也就是求出二者的内积。

说到回归，一般都是指 线性回归(linear regression)。线性回归意味着可以将输入项分别乘以一些常量，再将结果加起来得到输出。

回归 原理

1、线性回归

我们应该怎样从一大堆数据里求出回归方程呢？ 假定输入数据存放在矩阵 x 中，而回归系数存放在向量 w 中。那么对于给定的数据 X1，预测结果将会通过 Y = X1^T w 给出。现在的问题是，手里有一些 X 和对应的 y，怎样才能找到 w 呢？一个常用的方法就是找出使误差最小的 w 。这里的误差是指预测 y 值和真实 y 值之间的差值，使用该误差的简单累加将使得正差值和负差值相互抵消，所以我们采用平方误差。

平方误差可以写做:

$$\sum_{i=1}^{m}(y_i - x_i^T w)^2$$

用矩阵表示还可以写做 $(y-Xw)^T(y-Xw)$ 。如果对 w 求导，得到 $X^T(y-Xw)$ ，令其等于零，解出 w 如下（具体求导过程为: http://blog.csdn.net/nomadlx53/article/details/50849941 ）:

$$\hat{w} = (X^TX)^{-1}X^Ty$$

1.1、线性回归 须知概念

1.1.1、矩阵求逆

因为我们在计算回归方程的回归系数时，用到的计算公式如下:

$$\hat{w} = (X^TX)^{-1}X^Ty$$

需要对矩阵求逆，因此这个方程只在逆矩阵存在的时候适用，我们在程序代码中对此作出判断。 判断矩阵是否可逆的一个可选方案是:

判断矩阵的行列式是否为 0，若为 0 ，矩阵就不存在逆矩阵，不为 0 的话，矩阵才存在逆矩阵。

1.1.2、最小二乘法

最小二乘法（又称最小平方法）是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。

1.2、线性回归 工作原理

```
读入数据，将数据特征x、特征标签y存储在矩阵x、y中
验证 x^Tx 矩阵是否可逆
使用最小二乘法求得 回归系数 w 的最佳估计
```

1.3、线性回归 开发流程

```
收集数据：采用任意方法收集数据
准备数据：回归需要数值型数据，标称型数据将被转换成二值型数据
分析数据：绘出数据的可视化二维图将有助于对数据做出理解和分析，在采用缩减法求得新回归系数之后，可以将新拟合线绘在图上作为对比
训练算法：找到回归系数
测试算法：使用 R^2 或者预测值和数据的拟合度，来分析模型的效果
使用算法：使用回归，可以在给定输入的时候预测出一个数值，这是对分类方法的提升，因为这样可以预测连续型数据而不仅仅是离散的类别标签
```

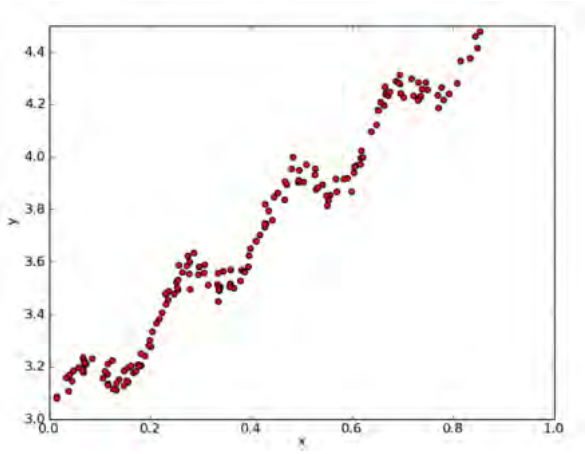
1.4、线性回归 算法特点

```
优点：结果易于理解，计算上不复杂。
缺点：对非线性的数据拟合不好。
适用于数据类型：数值型和标称型数据。
```

1.5、线性回归 项目案例

1.5.1、线性回归 项目概述

根据下图中的点，找出该数据的最佳拟合直线。



数据格式为:

```
x0          x1          y
1.000000        0.067732        3.176513
1.000000        0.427810        3.816464
1.000000        0.995731        4.550095
1.000000        0.738336        4.256571
```

1.5.2、线性回归 编写代码

```
def loadDataSet(fileName):
    """ 加载数据
        解析以tab键分隔的文件中的浮点数
    Returns:
        dataMat :   feature 对应的数据集
        labelMat :  feature 对应的分类标签，即类别标签

    """
    # 获取样本特征的总数，不算最后的目标变量
    numFeat = len(open(fileName).readline().split('\t')) - 1
    dataMat = []
    labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        # 读取每一行
        lineArr =[]
        # 删除一行中以tab分隔的数据前后的空白符号
        curLine = line.strip().split('\t')
        # i 从0到2，不包括2
        for i in range(numFeat):
            # 将数据添加到lineArr List中，每一行数据测试数据组成一个行向量
            lineArr.append(float(curLine[i]))
            # 将测试数据的输入数据部分存储到dataMat 的List中
        dataMat.append(lineArr)
        # 将每一行的最后一个数据，即类别，或者叫目标变量存储到labelMat List中
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat


def standRegres(xArr,yArr):
    '''
    Description:
        线性回归
    Args:
        xArr ：输入的样本数据，包含每个样本数据的 feature
        yArr ：对应于输入数据的类别标签，也就是每个样本对应的目标变量
    Returns:
        ws：回归系数
    '''

    # mat()函数将xArr，yArr转换为矩阵 mat().T 代表的是对矩阵进行转置操作
    xMat = mat(xArr)
    yMat = mat(yArr).T
    # 矩阵乘法的条件是左矩阵的列数等于右矩阵的行数
    xTx = xMat.T*xMat
    # 因为要用到xTx的逆矩阵，所以事先需要确定计算得到的xTx是否可逆，条件是矩阵的行列式不为0
    # linalg.det() 函数是用来求得矩阵的行列式的，如果矩阵的行列式为0，则这个矩阵是不可逆的，就无法进行接下来的运算
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    # 最小二乘法
    # http://www.apache.wiki/pages/viewpage.action?pageId=5505133
    # 书中的公式，求得w的最优解
    ws = xTx.I * (xMat.T*yMat)
    return ws


def regression1():
    xArr, yArr = loadDataSet("input/8.Regression/data.txt")
    xMat = mat(xArr)
    yMat = mat(yArr)
    ws = standRegres(xArr, yArr)
    fig = plt.figure()
    ax = fig.add_subplot(111)               #add_subplot(349)函数的参数的意思是，将画布分成3行4列图像画在从左到右从上到下第9块
    ax.scatter(xMat[:, 1].flatten(), yMat.T[:, 0].flatten().A[0]) #scatter 的x是xMat中的第二列，y是yMat的第一列
    xCopy = xMat.copy()
    xCopy.sort(0)
    yHat = xCopy * ws
    ax.plot(xCopy[:, 1], yHat)
    plt.show()
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/8.PredictiveNumericalDataRegression/regression.py

1.5.3、线性回归 拟合效果

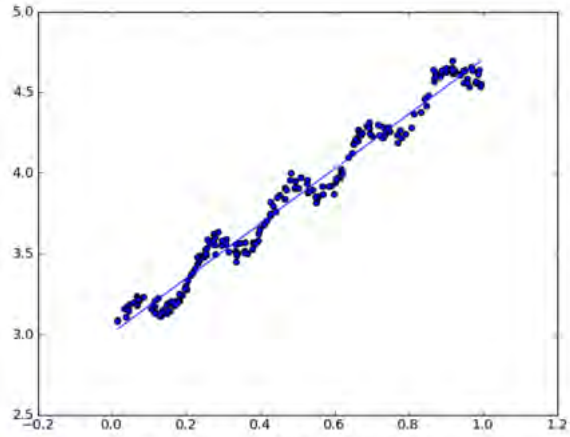


Figure 8.2 Data from ex0.txt with a best-fit line fitted to the data

2、局部加权线性回归

线性回归的一个问题是有可能出现欠拟合现象，因为它求的是具有最小均方差的无偏估计。显而易见，如果模型欠拟合将不能取得最好的预测效果。所以有些方法允许在估计中引入一些偏差，从而降低预测的均方误差。

一个方法是局部加权线性回归（Locally Weighted Linear Regression，LWLR）。在这个算法中，我们给预测点附近的每个点赋予一定的权重，然后与 线性回归 类似，在这个子集上基于最小均方误差来进行普通的回归。我们需要最小化的目标函数大致为:

$$\sum_i w(y^{(i)} - \hat{y}^{(i)})^2$$

与 kNN 一样，这种算法每次预测均需要事先选取出对应的数据子集。该算法解出回归系数 w 的形式如下:

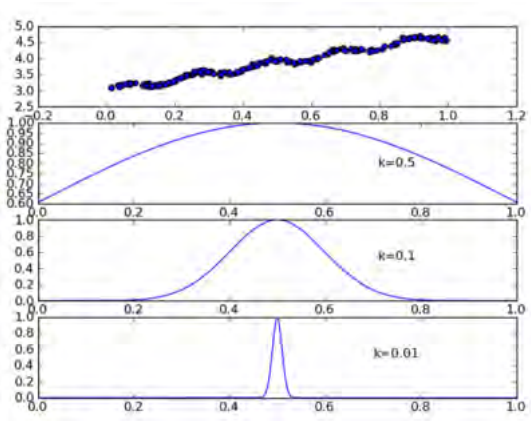
$$\hat{w} = (X^T W X)^{-1} X^T W y$$

其中 w 是一个矩阵，用来给每个数据点赋予权重。

LWLR 使用 “核”（与支持向量机中的核类似）来对附近的点赋予更高的权重。核的类型可以自由选择，最常用的核就是高斯核，高斯核对应的权重如下:

$$w(i,i) = \exp\left(\frac{|x^{(i)} - x|}{-2k^2}\right)$$

这样就构建了一个只含对角元素的权重矩阵 w，并且点 x 与 x(i) 越近，w(i, i) 将会越大。上述公式中包含一个需要用户指定的参数 k，它决定了对附近的点赋予多大的权重，这也是使用 LWLR 时唯一需要考虑的参数，下面的图给出了参数 k 与权重的关系。



上面的图是 每个点的权重图（假定我们正预测的点是 x = 0.5），最上面的图是原始数据集，第二个图显示了当 k = 0.5 时，大部分的数据都用于训练回归模型；而最下面的图显示当 k=0.01 时，仅有很少的局部点被用于训练回归模型。

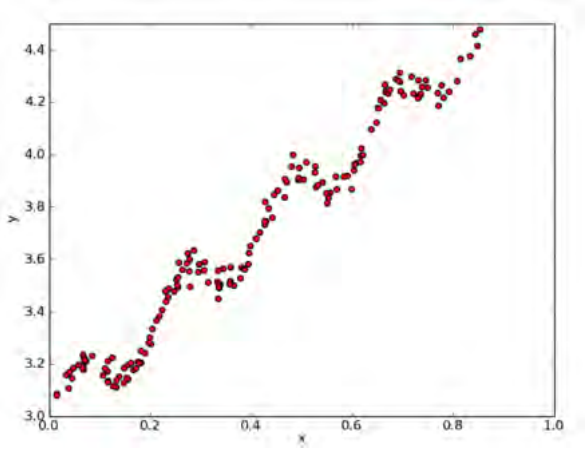
2.1、局部加权线性回归 工作原理

```
读入数据，将数据特征x、特征标签y存储在矩阵x、y中
利用高斯核构造一个权重矩阵 W，对预测点附近的点施加权重
验证 X^TWX 矩阵是否可逆
使用最小二乘法求得 回归系数 w 的最佳估计
```

2.2、局部加权线性回归 项目案例

2.2.1、局部加权线性回归 项目概述

我们仍然使用上面 线性回归 的数据集，对这些点进行一个 局部加权线性回归 的拟合。



数据格式为:

```
1.000000        0.067732        3.176513
1.000000        0.427810        3.816464
1.000000        0.995731        4.550095
1.000000        0.738336        4.256571
```

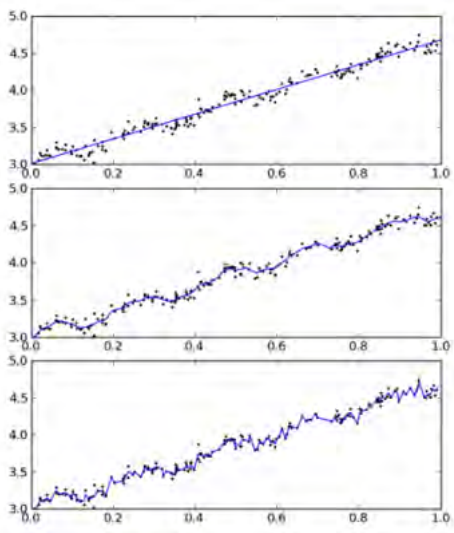
2.2.2、局部加权线性回归 编写代码

```
    # 局部加权线性回归
def lwlr(testPoint,xArr,yArr,k=1.0):
    '''
        Description:
            局部加权线性回归，在待预测点附近的每个点赋予一定的权重，在子集上基于最小均方差来进行普通的回归。
        Args:
            testPoint：样本点
            xArr：样本的特征数据，即 feature
            yArr：每个样本对应的类别标签，即目标变量
            k:关于赋予权重矩阵的核的一个参数，与权重的衰减速率有关
        Returns:
            testPoint * ws：数据点与具有权重的系数相乘得到的预测点
        Notes:
            这其中会用到计算权重的公式，w = e^((x^((i))-x) / -2k^2)
            理解：x为某个预测点，x^((i))为样本点，样本点距离预测点越近，贡献的误差越大（权值越大），越远则贡献的误差越小（权值越小）。
            关于预测点的选取，在我的代码中取的是样本点。其中k是带宽参数，控制w（钟形函数）的宽窄程度，类似于高斯函数的标准差。
            算法思路：假设预测点取样本点中的第i个样本点（共m个样本点），遍历1到m个样本点（含第i个），算出每一个样本点与预测点的距离，
            也就可以计算出每个样本贡献误差的权值，可以看出w是一个有m个元素的向量（写成对角阵形式）。
    '''
    # mat() 函数是将array转换为矩阵的函数， mat().T 是转换为矩阵之后，再进行转置操作
    xMat = mat(xArr)
    yMat = mat(yArr).T
    # 获得xMat矩阵的行数
    m = shape(xMat)[0]
    # eye()返回一个对角线元素为1，其他元素为0的二维数组，创建权重矩阵weights，该矩阵为每个样本点初始化了一个权重
    weights = mat(eye((m)))
    for j in range(m):
        # testPoint 的形式是 一个行向量的形式
        # 计算 testPoint 与输入样本点之间的距离，然后下面计算出每个样本贡献误差的权值
        diffMat = testPoint - xMat[j,:]
        # k控制衰减的速度
        weights[j,j] = exp(diffMat*diffMat.T/(-2.0*k**2))
    # 根据矩阵乘法计算 xTx ，其中的 weights 矩阵是样本点对应的权重矩阵
    xTx = xMat.T * (weights * xMat)
    if linalg.det(xTx) == 0.0:
        print ("This matrix is singular, cannot do inverse")
        return
    # 计算出回归系数的一个估计
    ws = xTx.I * (xMat.T * (weights * yMat))
    return testPoint * ws

def lwlrTest(testArr,xArr,yArr,k=1.0):
    '''
        Description:
            测试局部加权线性回归，对数据集中每个点调用 lwlr() 函数
        Args:
            testArr：测试所用的所有样本点
            xArr：样本的特征数据，即 feature
            yArr：每个样本对应的类别标签，即目标变量
            k：控制核函数的衰减速率
        Returns:
            yHat：预测点的估计值
    '''
    # 得到样本点的总数
```

```
    m = shape(testArr)[0]
    # 构建一个全部都是 0 的 1 * m 的矩阵
    yHat = zeros(m)
    # 循环所有的数据点，并将lwlr运用于所有的数据点
    for i in range(m):
        yHat[i] = lwlr(testArr[i],xArr,yArr,k)
    # 返回估计值
    return yHat

def lwlrTestPlot(xArr,yArr,k=1.0):
    '''
        Description:
            首先将 X 排序，其余的都与lwlrTest相同，这样更容易绘图
        Args:
            xArr：样本的特征数据，即 feature
            yArr：每个样本对应的类别标签，即目标变量，实际值
            k：控制核函数的衰减速率的有关参数，这里设定的是常量值 1
        Return:
            yHat：样本点的估计值
            xCopy：xArr的复制
    '''
    # 生成一个与目标变量数目相同的 0 向量
    yHat = zeros(shape(yArr))
    # 将 xArr 转换为 矩阵形式
    xCopy = mat(xArr)
    # 排序
    xCopy.sort(0)
    # 开始循环，为每个样本点进行局部加权线性回归，得到最终的目标变量估计值
    for i in range(shape(xArr)[0]):
        yHat[i] = lwlr(xCopy[i],xArr,yArr,k)
    return yHat,xCopy


#test for LWLR
def regression2():
    xArr, yArr = loadDataSet("input/8.Regression/data.txt")
    yHat = lwlrTest(xArr, xArr, yArr, 0.003)
    xMat = mat(xArr)
    srtInd = xMat[:,1].argsort(0)           #argsort()函数是将x中的元素从小到大排列，提取其对应的index(索引)，然后输出
    xSort=xMat[srtInd][:,0,:]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(xSort[:,1], yHat[srtInd])
    ax.scatter(xMat[:,1].flatten().A[0], mat(yArr).T.flatten().A[0] , s=2, c='red')
    plt.show()
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/8.PredictiveNumericalDataRegression/regression.py

2.2.3、局部加权线性回归 拟合效果



上图使用了 3 种不同平滑值绘出的局部加权线性回归的结果。上图中的平滑系数 k =1.0，中图 k = 0.01，下图 k = 0.003 。可以看到，k = 1.0 时的模型效果与最小二乘法差不多，k=0.01时该模型可以挖出数据的潜在规律，而 k=0.003时则考虑了太多的噪声，进而导致了过拟合现象。

2.3、局部加权线性回归 注意事项

局部加权线性回归也存在一个问题，即增加了计算量，因为它对每个点做预测时都必须使用整个数据集。

3、线性回归 & 局部加权线性回归 项目案例

到此为止，我们已经介绍了找出最佳拟合直线的两种方法，下面我们用这些技术来预测鲍鱼的年龄。

3.1、项目概述

我们有一份来自 UCI 的数据集合的数据，记录了鲍鱼（一种介壳类水生动物）的年龄。鲍鱼年龄可以从鲍鱼壳的层数推算得到。

3.2、开发流程

```
收集数据：采用任意方法收集数据
准备数据：回归需要数值型数据，标称型数据将被转换成二值型数据
分析数据：绘出数据的可视化二维图将有助于对数据做出理解和分析，在采用缩减法求得新回归系数之后，可以将新拟合线绘在图上作为对比
训练算法：找到回归系数
测试算法：使用 rssError()函数 计算预测误差的大小，来分析模型的效果
使用算法：使用回归，可以在给定输入的时候预测出一个数值，这是对分类方法的提升，因为这样可以预测连续型数据而不仅仅是离散的类别标签
```

> 收集数据: 采用任意方法收集数据

> 准备数据: 回归需要数值型数据，标称型数据将被转换成二值型数据

数据存储格式:

```
1       0.455   0.365   0.095   0.514   0.2245  0.101   0.15    15
1       0.35    0.265   0.09    0.2255  0.0995  0.0485  0.07    7
-1      0.53    0.42    0.135   0.677   0.2565  0.1415  0.21    9
1       0.44    0.365   0.125   0.516   0.2155  0.114   0.155   10
0       0.33    0.255   0.08    0.205   0.0895  0.0395  0.055   7
```

> 分析数据: 绘出数据的可视化二维图将有助于对数据做出理解和分析，在采用缩减法求得新回归系数之后，可以将新拟合线绘在图上作为对比

> 训练算法: 找到回归系数

使用上面我们讲到的 局部加权线性回归 训练算法，求出回归系数

> 测试算法: 使用 rssError()函数 计算预测误差的大小，来分析模型的效果

```
# test for abloneDataSet
def abaloneTest():
    '''
    Desc:
        预测鲍鱼的年龄
    Args:
        None
    Returns:
        None
    '''
    # 加载数据
    abX, abY = loadDataSet("input/8.Regression/abalone.txt")
    # 使用不同的核进行预测
    oldyHat01 = lwlrTest(abX[0:99], abX[0:99], abY[0:99], 0.1)
    oldyHat1 = lwlrTest(abX[0:99], abX[0:99], abY[0:99], 1)
    oldyHat10 = lwlrTest(abX[0:99], abX[0:99], abY[0:99], 10)
    # 打印出不同的核预测值与训练数据集上的真实值之间的误差大小
    print "old yHat01 error Size is :" , rssError(abY[0:99], oldyHat01.T)
    print "old yHat1 error Size is :" , rssError(abY[0:99], oldyHat1.T)
    print "old yHat10 error Size is :" , rssError(abY[0:99], oldyHat10.T)

    # 打印出 不同的核预测值 与 新数据集（测试数据集）上的真实值之间的误差大小
    newyHat01 = lwlrTest(abX[100:199], abX[0:99], abY[0:99], 0.1)
    print "new yHat01 error Size is :" , rssError(abY[0:99], newyHat01.T)
    newyHat1 = lwlrTest(abX[100:199], abX[0:99], abY[0:99], 1)
    print "new yHat1 error Size is :" , rssError(abY[0:99], newyHat1.T)
    newyHat10 = lwlrTest(abX[100:199], abX[0:99], abY[0:99], 10)
    print "new yHat10 error Size is :" , rssError(abY[0:99], newyHat10.T)

    # 使用简单的 线性回归 进行预测，与上面的计算进行比较
    standWs = standRegres(abX[0:99], abY[0:99])
    standyHat = mat(abX[100:199]) * standWs
    print "standRegress error Size is:", rssError(abY[100:199], standyHat.T.A)
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/8.PredictiveNumericalDataRegression/regression.py

根据我们上边的测试，可以看出:

简单线性回归达到了与局部加权现行回归类似的效果。这也说明了一点，必须在未知数据上比较效果才能选取到最佳模型。那么最佳的核大小是 10 吗？或许是，但如果想得到更好的效果，应该用 10 个不同的样本集做 10 次测试来比较结果。

> 使用算法: 使用回归，可以在给定输入的时候预测出一个数值，这是对分类方法的提升，因为这样可以预测连续型数据而不仅仅是离散的类别标签

4、缩减系数来“理解”数据

如果数据的特征比样本点还多应该怎么办？是否还可以使用线性回归和之前的方法来做预测？答案是否定的，即我们不能再使用前面介绍的方法。这是因为在计算 $(\mathbf{X}^T\mathbf{X})^{-1}$ 的时候会出错。

如果特征比样本点还多(n > m)，也就是说输入数据的矩阵 x 不是满秩矩阵。非满秩矩阵求逆时会出现问题。

为了解决这个问题，我们引入了 `岭回归（ridge regression）` 这种缩减方法。接着是 `lasso法`，最后介绍 `前向逐步回归`。

4.1、岭回归

简单来说，岭回归就是在矩阵 $\mathbf{X^TX}$ 上加一个 λI 从而使得矩阵非奇异，进而能对 $\mathbf{X^TX} + \lambda\mathbf{I}$ 求逆。其中矩阵I是一个 m * m 的单位矩阵， 对角线上元素全为 1，其他元素全为0。而λ是一个用户定义的数值，后面会做介绍。在这种情况下，回归系数的计算公式将变成：

$$\hat{w} = (X^TX + \lambda I)^{-1}X^Ty$$

岭回归最先用来处理特征数多于样本数的情况，现在也用于在估计中加入偏差，从而得到更好的估计。这里通过引入 λ 来限制了所有 w 之和，通过引入该惩罚项，能够减少不重要的参数，这个技术在统计学中也叫作 `缩减(shrinkage)`。

对于有些矩阵，矩阵中某个元素的一个很小的变动，会引起最后计算结果误差很大，这种矩阵称为"病态矩阵"。有些时候不正确的计算方法也会使一个正常的矩阵在运算中表现出病态。对于高斯消去法来说，如果主元（即对角线上的元素）上的元素很小，在计算时就会表现出病态的特征。

回归分析中常用的最小二乘法是一种无偏估计。对于一个适定问题，X通常是列满秩的

$$X\theta = y$$

采用最小二乘法，定义损失函数为残差的平方，最小化损失函数

$$||X\theta - y||^2$$

上述优化问题可以采用梯度下降法进行求解，也可以采用如下公式进行直接求解

$$\theta = (X^TX)^{-1}X^Ty$$

当X不是列满秩时，或者某些列之间的线性相关性比较大时， X^TX 的行列式接近于0，即 X^TX 接近于奇异，上述问题变为一个不适定问题，此时，计算 $(X^TX)^{-1}$ 时误差会很大，传统的最小二乘法缺乏稳定性与可靠性。

为了解决上述问题，我们需要将不适定问题转化为适定问题：我们为上述损失函数加上一个正则化项，变为

$$||X\theta - y||^2 + ||\Gamma\theta||^2$$

其中，我们定义 $\Gamma = \alpha I$ 。

于是：

$$\theta(\alpha) = (X^TX + \alpha I)^{-1}X^Ty$$

（sitar alpha处多了一个右括号）

上式中， I 是单位矩阵。

随着 α 的增大， $\theta(\alpha)$ 各元素 $\theta(\alpha)_i$ 的绝对值均趋于不断变小，它们相对于正确值 θ_i 的偏差也越来越大。 α 趋于无穷大时， $\theta(\alpha)$ 趋于0。其中， $\theta(\alpha)$ 随 α 的改变而变化的轨迹，就称为岭迹。实际计算中可选非常多的 α 值，做出一个岭迹图，看看这个图在取哪个值的时候变稳定了，那就确定 α 值了。

岭回归是对最小二乘回归的一种补充，它损失了无偏性，来换取高的数值稳定性，从而得到较高的计算精度。

缩减方法可以去掉不重要的参数，因此能更好地理解数据。此外，与简单的线性回归相比，缩减法能取得更好的预测效果。

这里通过预测误差最小化得到 λ: 数据获取之后，首先抽一部分数据用于测试，剩余的作为训练集用于训练参数 w。训练完毕后在测试集上测试预测性能。通过选取不同的 λ 来重复上述测试过程，最终得到一个使预测误差最小的 λ 。

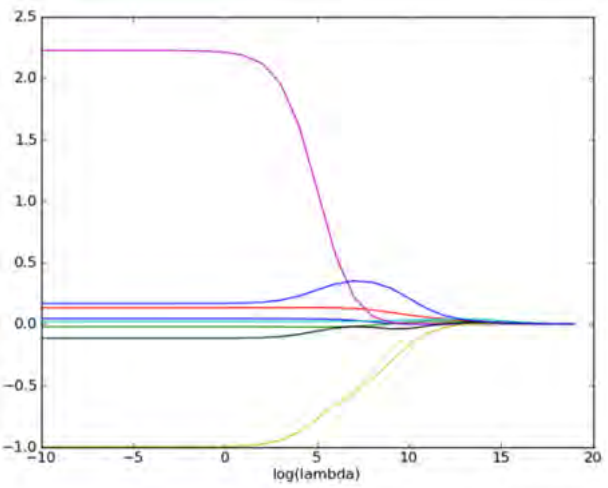
4.1.1、岭回归 原始代码

```
def ridgeRegres(xMat,yMat,lam=0.2):
    '''
        Desc:
            这个函数实现了给定 lambda 下的岭回归求解。
            如果数据的特征比样本点还多，就不能再使用上面介绍的的线性回归和局部现行回归了，因为计算 (xTx)^(-1)会出现错误。
            如果特征比样本点还多（n > m），也就是说，输入数据的矩阵x不是满秩矩阵。非满秩矩阵在求逆时会出现问题。
            为了解决这个问题，我们下边讲一下：岭回归，这是我们要讲的第一种缩减方法。
        Args:
            xMat：样本的特征数据，即 feature
            yMat：每个样本对应的类别标签，即目标变量，实际值
            lam：引入的一个λ值，使得矩阵非奇异
        Returns:
            经过岭回归公式计算得到的回归系数
    '''

    xTx = xMat.T*xMat
    # 岭回归就是在矩阵 xTx 上加一个 λI 从而使得矩阵非奇异，进而能对 xTx + λI 求逆
    denom = xTx + eye(shape(xMat)[1])*lam
    # 检查行列式是否为零，即矩阵是否可逆，行列式为0的话就不可逆，不为0的话就是可逆。
```

```
    if linalg.det(denom) == 0.0:
        print ("This matrix is singular, cannot do inverse")
        return
    ws = denom.I * (xMat.T*yMat)
    return ws


def ridgeTest(xArr,yArr):
    '''
        Desc:
            函数 ridgeTest() 用于在一组 λ 上测试结果
        Args:
            xArr：样本数据的特征，即 feature
            yArr：样本数据的类别标签，即真实数据
        Returns:
            wMat：将所有的回归系数输出到一个矩阵并返回
    '''

    xMat = mat(xArr)
    yMat=mat(yArr).T
    # 计算Y的均值
    yMean = mean(yMat,0)
    # Y的所有的特征减去均值
    yMat = yMat - yMean
    # 标准化 x，计算 xMat 平均值
    xMeans = mean(xMat,0)
    # 然后计算 X的方差
    xVar = var(xMat,0)
    # 所有特征都减去各自的均值并除以方差
    xMat = (xMat - xMeans)/xVar
    # 可以在 30 个不同的 lambda 下调用 ridgeRegres() 函数。
    numTestPts = 30
    # 创建30 * m 的全部数据为0 的矩阵
    wMat = zeros((numTestPts,shape(xMat)[1]))
    for i in range(numTestPts):
        # exp() 返回 e^x
        ws = ridgeRegres(xMat,yMat,exp(i-10))
        wMat[i,:]=ws.T
    return wMat


#test for ridgeRegression
def regression3():
    abX,abY = loadDataSet("input/8.Regression/abalone.txt")
    ridgeWeights = ridgeTest(abX, abY)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(ridgeWeights)
    plt.show()
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/8.PredictiveNumericalDataRegression/regression.py

4.1.2、岭回归在鲍鱼数据集上的运行效果



上图绘制除了回归系数与 log(λ) 的关系。在最左边，即 λ 最小时，可以得到所有系数的原始值（与线性回归一致）；而在右边，系数全部缩减为0；在中间部分的某值将可以取得最好的预测效果。为了定量地找到最佳参数值，还需要进行交叉验证。另外，要判断哪些变量对结果预测最具有影响力，在上图中观察它们对应的系数大小就可以了。

4.2、lasso

在增加如下约束时，普通的最小二乘法回归会得到与岭回归一样的公式:

$$\sum_{k=1}^{n} w_k^2 \leq \lambda$$

上式限定了所有回归系数的平方和不能大于 λ 。使用普通的最小二乘法回归在当两个或更多的特征相关时，可能会得到一个很大的正系数和一个很大的负系数。正式因为上述限制条件的存在，使用岭回归可以避免这个问题。

与岭回归类似，另一个缩减方法lasso也对回归系数做了限定，对应的约束条件如下:

$$\sum_{k=1}^{n} |w_k| \leq \lambda$$

唯一的不同点在于，这个约束条件使用绝对值取代了平方和。虽然约束形式只是稍作变化，结果却大相径庭: 在 λ 足够小的时候，一些系数会因此被迫缩减到 0.这个特性可以帮助我们更好地理解数据。

4.3、前向逐步回归

前向逐步回归算法可以得到与 lasso 差不多的效果，但更加简单。它属于一种贪心算法，即每一步都尽可能减少误差。一开始，所有权重都设置为 1，然后每一步所做的决策是对某个权重增加或减少一个很小的值。

伪代码如下:

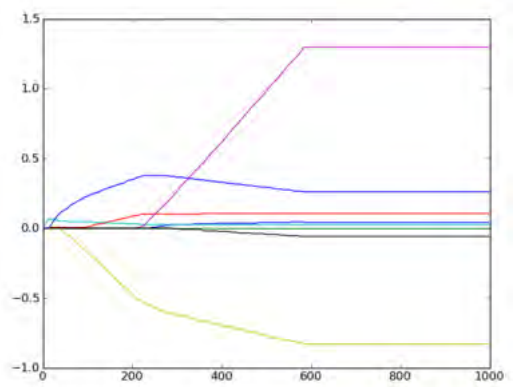
```
数据标准化，使其分布满足 0 均值 和单位方差
在每轮迭代过程中:
    设置当前最小误差 lowestError 为正无穷
    对每个特征:
        增大或缩小:
            改变一个系数得到一个新的 w
            计算新 w 下的误差
            如果误差 Error 小于当前最小误差 lowestError: 设置 Wbest 等于当前的 W
        将 W 设置为新的 Wbest
```

4.3.1、前向逐步回归 原始代码

```
def stageWise(xArr,yArr,eps=0.01,numIt=100):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean     # 也可以规则化ys但会得到更小的coef
    xMat = regularize(xMat)
    m,n=shape(xMat)
    #returnMat = zeros((numIt,n)) # 测试代码删除
    ws = zeros((n,1)); wsTest = ws.copy(); wsMax = ws.copy()
    for i in range(numIt):
        print (ws.T)
        lowestError = inf;
        for j in range(n):
            for sign in [-1,1]:
                wsTest = ws.copy()
                wsTest[j] += eps*sign
                yTest = xMat*wsTest
                rssE = rssError(yMat.A,yTest.A)
                if rssE < lowestError:
                    lowestError = rssE
                    wsMax = wsTest
        ws = wsMax.copy()
        returnMat[i,:]=ws.T
    return returnMat


#test for stageWise
def regression4():
    xArr,yArr=loadDataSet("input/8.Regression/abalone.txt")
    stageWise(xArr,yArr,0.01,200)
    xMat = mat(xArr)
    yMat = mat(yArr).T
    xMat = regularize(xMat)
    yM = mean(yMat,0)
    yMat = yMat - yM
    weights = standRegres(xMat, yMat.T)
    print (weights.T)
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/8.PredictiveNumericalDataRegression/regression.py

4.3.2、逐步线性回归在鲍鱼数据集上的运行效果



逐步线性回归算法的主要优点在于它可以帮助人们理解现有的模型并作出改进。当构建了一个模型后，可以运行该算法找出重要的特征，这样就有可能及时停止对那些不重要特征的手机。最后，如果用于测试，该算法每100次迭代后就可以构建出一个模型，可以使用类似于10折交叉验证的方法比较这些模型，最终选择使误差最小的模型。

4.4、小结

当应用缩减方法（如逐步线性回归或岭回归）时，模型也就增加了偏差（bias），与此同时却减小了模型的方差。

5、回归 项目案例

项目案例1: 预测乐高玩具套装的价格

项目概述

Dangler 喜欢为乐高套装估价，我们用回归技术来帮助他建立一个预测模型。

开发流程

```
(1) 收集数据：用 Google Shopping 的API收集数据。
(2) 准备数据：从返回的JSON数据中抽取价格。
(3) 分析数据：可视化并观察数据。
(4) 训练算法：构建不同的模型，采用逐步线性回归和直接的线性回归模型。
(5) 测试算法：使用交叉验证来测试不同的模型，分析哪个效果最好。
(6) 使用算法：这次练习的目标就是生成数据模型。
```

> 收集数据: 使用 Google 购物的 API

由于 Google 提供的 api 失效，我们只能自己下载咯，将数据存储在了 input 文件夹下的 setHtml 文件夹下

> 准备数据: 从返回的 JSON 数据中抽取价格

因为我们这里不是在线的，就不再是 JSON 了，我们直接解析线下的网页，得到我们想要的数据。

> 分析数据: 可视化并观察数据

这里我们将解析得到的数据打印出来，然后观察数据。

> 训练算法: 构建不同的模型

```
from numpy import *
from bs4 import BeautifulSoup

# 从页面读取数据，生成retX和retY列表
def scrapePage(retX, retY, inFile, yr, numPce, origPrc):

    # 打开并读取HTML文件
    fr = open(inFile)
    soup = BeautifulSoup(fr.read())
    i=1

    # 根据HTML页面结构进行解析
    currentRow = soup.findAll('table', r="%d" % i)
    while(len(currentRow)!=0):
        currentRow = soup.findAll('table', r="%d" % i)
        title = currentRow[0].findAll('a')[1].text
        lwrTitle = title.lower()

        # 查找是否有全新标签
        if (lwrTitle.find('new') > -1) or (lwrTitle.find('nisb') > -1):
            newFlag = 1.0
        else:
            newFlag = 0.0

        # 查找是否已经标志出售，我们只收集已出售的数据
        soldUnicde = currentRow[0].findAll('td')[3].findAll('span')
```

```
        if len(soldUnicde)==0:
            print "item #%d did not sell" % i
        else:
            # 解析页面获取当前价格
            soldPrice = currentRow[0].findAll('td')[4]
            priceStr = soldPrice.text
            priceStr = priceStr.replace('$','') #strips out $
            priceStr = priceStr.replace(',','') #strips out ,
            if len(soldPrice)>1:
                priceStr = priceStr.replace('Free shipping', '')
            sellingPrice = float(priceStr)

            # 去掉不完整的套装价格
            if  sellingPrice > origPrc * 0.5:
                    print "%d\t%d\t%d\t%f\t%f" % (yr,numPce,newFlag,origPrc, sellingPrice)
                    retX.append([yr, numPce, newFlag, origPrc])
                    retY.append(sellingPrice)
        i += 1
        currentRow = soup.findAll('table', r="%d" % i)

# 依次读取六种乐高套装的数据，并生成数据矩阵
def setDataCollect(retX, retY):
    scrapePage(retX, retY, 'input/8.Regression/setHtml/lego8288.html', 2006, 800, 49.99)
    scrapePage(retX, retY, 'input/8.Regression/setHtml/lego10030.html', 2002, 3096, 269.99)
    scrapePage(retX, retY, 'input/8.Regression/setHtml/lego10179.html', 2007, 5195, 499.99)
    scrapePage(retX, retY, 'input/8.Regression/setHtml/lego10181.html', 2007, 3428, 199.99)
    scrapePage(retX, retY, 'input/8.Regression/setHtml/lego10189.html', 2008, 5922, 299.99)
    scrapePage(retX, retY, 'input/8.Regression/setHtml/lego10196.html', 2009, 3263, 249.99)
```

> 测试算法：使用交叉验证来测试不同的模型，分析哪个效果最好

```
# 交叉验证测试岭回归
def crossValidation(xArr,yArr,numVal=10):
    # 获得数据点个数，xArr和yArr具有相同长度
    m = len(yArr)
    indexList = range(m)
    errorMat = zeros((numVal,30))

    # 主循环 交叉验证循环
    for i in range(numVal):
        # 随机拆分数据，将数据分为训练集（90%）和测试集（10%）
        trainX=[]; trainY=[]
        testX = []; testY = []

        # 对数据进行混洗操作
        random.shuffle(indexList)

        # 切分训练集和测试集
        for j in range(m):
            if j < m*0.9:
                trainX.append(xArr[indexList[j]])
                trainY.append(yArr[indexList[j]])
            else:
                testX.append(xArr[indexList[j]])
                testY.append(yArr[indexList[j]])

        # 获得回归系数矩阵
        wMat = ridgeTest(trainX,trainY)

        # 循环遍历矩阵中的30组回归系数
        for k in range(30):
            # 读取训练集和数据集
            matTestX = mat(testX); matTrainX=mat(trainX)
            # 对数据进行标准化
            meanTrain = mean(matTrainX,0)
            varTrain = var(matTrainX,0)
            matTestX = (matTestX-meanTrain)/varTrain

            # 测试回归效果并存储
            yEst = matTestX * mat(wMat[k,:]).T + mean(trainY)

            # 计算误差
            errorMat[i,k] = ((yEst.T.A-array(testY))**2).sum()

    # 计算误差估计值的均值
    meanErrors = mean(errorMat,0)
    minMean = float(min(meanErrors))
    bestWeights = wMat[nonzero(meanErrors==minMean)]

    # 不要使用标准化的数据，需要对数据进行还原来得到输出结果
    xMat = mat(xArr); yMat=mat(yArr).T
    meanX = mean(xMat,0); varX = var(xMat,0)
```

```
    unReg = bestWeights/varX

    # 输出构建的模型
    print "the best model from Ridge Regression is:\n",unReg
    print "with constant term: ",-1*sum(multiply(meanX,unReg)) + mean(yMat)


# predict for lego's price
def regression5():
    lgX = []
    lgY = []

    setDataCollect(lgX, lgY)
    crossValidation(lgX, lgY, 10)
```

> 使用算法：这次练习的目标就是生成数据模型

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/8.PredictiveNumericalDataRegression/regression.py

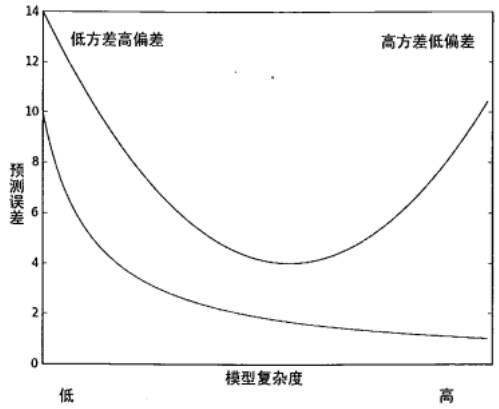
6、附加 权衡偏差和方差

任何时候，一旦发现模型和测量值之间存在差异，就说出现了误差。当考虑模型中的 “噪声” 或者说误差时，必须考虑其来源。你可能会对复杂的过程进行简化，这将导致在模型和测量值之间出现 “噪声” 或误差，若无法理解数据的真实生成过程，也会导致差异的产生。另外，测量过程本身也可能产生 “噪声” 或者问题。下面我们举一个例子，我们使用 线性回归 和 局部加权线性回归 处理过一个从文件导入的二维数据。

$$y = 3.0 + 1.7x + 0.1\sin(30x)+0.06N(0,1)$$

其中的 N(0, 1) 是一个均值为 0、方差为 1 的正态分布。我们尝试过禁用一条直线来拟合上述数据。不难想到，直线所能得到的最佳拟合应该是 3.0+1.7x 这一部分。这样的话，误差部分就是 0.1sin(30x)+0.06N(0, 1) 。在上面，我们使用了局部加权线性回归来试图捕捉数据背后的结构。该结构拟合起来有一定的难度，因此我们测试了多组不同的局部权重来找到具有最小测试误差的解。

下图给出了训练误差和测试误差的曲线图，上面的曲面就是测试误差，下面的曲线是训练误差。我们根据 预测鲍鱼年龄 的实验知道: 如果降低核的大小，那么训练误差将变小。从下图开看，从左到右就表示了核逐渐减小的过程。



一般认为，上述两种误差由三个部分组成: 偏差、测量误差和随机噪声。局部加权线性回归 和 预测鲍鱼年龄 中，我们通过引入了三个越来越小的核来不断增大模型的方差。

在缩减系数来“理解”数据这一节中，我们介绍了缩减法，可以将一些系数缩减成很小的值或直接缩减为 0 ，这是一个增大模型偏差的例子。通过把一些特征的回归系数缩减到 0 ，同时也就减小了模型的复杂度。例子中有 8 个特征，消除其中两个后不仅使模型更易理解，同时还降低了预测误差。对照上图，左侧是参数缩减过于严厉的结果，而右侧是无缩减的效果。

方差是可以度量的。如果从鲍鱼数据中取一个随机样本集（例如取其中 100 个数据）并用线性模型拟合，将会得到一组回归系数。同理，再取出另一组随机样本集并拟合，将会得到另一组回归系数。这些系数间的差异大小也就是模型方差的反映。

第9章 树回归

树回归 概述

我们本章介绍 CART(Classification And Regression Trees, 分类回归树) 的树构建算法。该算法既可以用于分类还可以用于回归。

树回归 场景

我们在第 8 章中介绍了线性回归的一些强大的方法，但这些方法创建的模型需要拟合所有的样本点（局部加权线性回归除外）。当数据拥有众多特征并且特征之间关系十分复杂时，构建全局模型的想法就显得太难了，也略显笨拙。而且，实际生活中很多问题都是非线性的，不可能使用全局线性模型来拟合任何数据。

一种可行的方法是将数据集切分成很多份易建模的数据，然后利用我们的线性回归技术来建模。如果首次切分后仍然难以拟合线性模型就继续切分。在这种切分方式下，树回归和回归法就相当有用。

除了我们在 第3章 中介绍的 决策树算法，我们介绍一个新的叫做 CART(Classification And Regression Trees, 分类回归树) 的树构建算法。该算法既可以用于分类还可以用于回归。

1、树回归 原理

1.1、树回归 原理概述

为成功构建以分段常数为叶节点的树，需要度量出数据的一致性。第3章使用树进行分类，会在给定节点时计算数据的混乱度。那么如何计算连续型数值的混乱度呢？

在这里，计算连续型数值的混乱度是非常简单的。首先计算所有数据的均值，然后计算每条数据的值到均值的差值。为了对正负差值同等看待，一般使用绝对值或平方值来代替上述差值。

上述做法有点类似于前面介绍过的统计学中常用的方差计算。唯一不同就是，方差是平方误差的均值(均方差)，而这里需要的是平方误差的总值(总方差)。总方差可以通过均方差乘以数据集中样本点的个数来得到。

1.2、树构建算法 比较

我们在 第3章 中使用的树构建算法是 ID3 。ID3 的做法是每次选取当前最佳的特征来分割数据，并按照该特征的所有可能取值来切分。也就是说，如果一个特征有 4 种取值，那么数据将被切分成 4 份。一旦按照某特征切分后，该特征在之后的算法执行过程中将不会再起作用，所以有观点认为这种切分方式过于迅速。另外一种方法是二分切分法，即每次把数据集切分成两份。如果数据的某特征值等于切分所要求的值，那么这些数据就进入树的左子树，反之则进入树的右子树。

除了切分过于迅速外， ID3 算法还存在另一个问题，它不能直接处理连续型特征。只有事先将连续型特征转换成离散型，才能在 ID3 算法中使用。但这种转换过程会破坏连续型变量的内在性质。而使用二元切分法则易于对树构造过程进行调整以处理连续型特征。具体的处理方法是: 如果特征值大于给定值就走左子树，否则就走右子树。另外，二分切分法也节省了树的构建时间，但这点意义也不是特别大，因为这些树构建一般是离线完成，时间并非需要重点关注的因素。

CART 是十分著名且广泛记载的树构建算法，它使用二元切分来处理连续型变量。对 CART 稍作修改就可以处理回归问题。第 3 章中使用香农熵来度量集合的无组织程度。如果选用其他方法来代替香农熵，就可以使用树构建算法来完成回归。

回归树与分类树的思路类似，但是叶节点的数据类型不是离散型，而是连续型。

1.2.1、附加 各常见树构造算法的划分分支方式

还有一点要说明，构建决策树算法，常用到的是三个方法: ID3, C4.5, CART.
三种方法区别是划分树的分支的方式:

1. ID3 是信息增益分支
2. C4.5 是信息增益率分支
3. CART 是 GINI 系数分支

工程上总的来说:

CART 和 C4.5 之间主要差异在于分类结果上，CART 可以回归分析也可以分类，C4.5 只能做分类；C4.5 子节点是可以多分的，而 CART 是无数个二叉子节点；

以此拓展出以 CART 为基础的 “树群” Random forest ， 以 回归树 为基础的 “树群” GBDT 。

1.3、树回归 工作原理

1、找到数据集切分的最佳位置，函数 chooseBestSplit() 伪代码大致如下:

```
对每个特征:
    对每个特征值:
        将数据集切分成两份（小于该特征值的数据样本放在左子树，否则放在右子树）
        计算切分的误差
        如果当前误差小于当前最小误差，那么将当前切分设定为最佳切分并更新最小误差
返回最佳切分的特征和阈值
```

2、树构建算法，函数 createTree() 伪代码大致如下:

```
找到最佳的待切分特征:
    如果该节点不能再分，将该节点存为叶节点
    执行二元切分
    在右子树调用 createTree() 方法
    在左子树调用 createTree() 方法
```

1.4、树回归 开发流程

```
(1) 收集数据：采用任意方法收集数据。
(2) 准备数据：需要数值型数据，标称型数据应该映射成二值型数据。
(3) 分析数据：绘出数据的二维可视化显示结果，以字典方式生成树。
(4) 训练算法：大部分时间都花费在叶节点树模型的构建上。
(5) 测试算法：使用测试数据上的R^2值来分析模型的效果。
(6) 使用算法：使用训练处的树做预测，预测结果还可以用来做很多事情。
```

1.5、树回归 算法特点

```
优点：可以对复杂和非线性的数据建模。
缺点：结果不易理解。
适用数据类型：数值型和标称型数据。
```

1.6、回归树 项目案例

1.6.1、项目概述

在简单数据集上生成一棵回归树。

1.6.2、开发流程

```
收集数据：采用任意方法收集数据
准备数据：需要数值型数据，标称型数据应该映射成二值型数据
分析数据：绘出数据的二维可视化显示结果，以字典方式生成树
训练算法：大部分时间都花费在叶节点树模型的构建上
测试算法：使用测试数据上的R^2值来分析模型的效果
使用算法：使用训练出的树做预测，预测结果还可以用来做很多事情
```

> 收集数据：采用任意方法收集数据

data1.txt 文件中存储的数据格式如下:

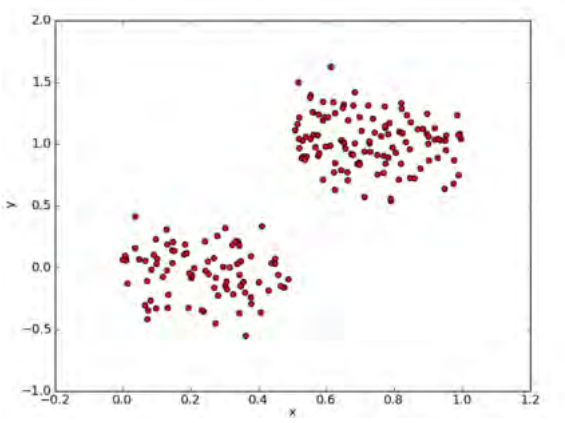
```
0.036098        0.155096
0.993349        1.077553
0.530897        0.893462
0.712386        0.564858
```

```
0.343554        -0.371700
0.098016        -0.332760
```

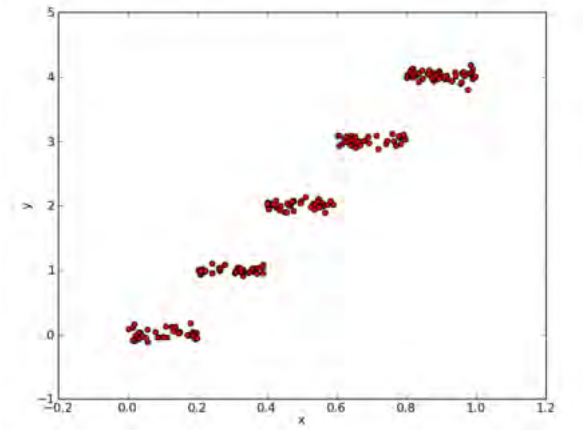
> 准备数据：需要数值型数据，标称型数据应该映射成二值型数据

> 分析数据：绘出数据的二维可视化显示结果，以字典方式生成树

基于 CART 算法构建回归树的简单数据集



用于测试回归树的分段常数数据集



> 训练算法: 构造树的数据结构

```
def binSplitDataSet(dataSet, feature, value):
    """binSplitDataSet(将数据集，按照feature列的value进行 二元切分)
        Description：在给定特征和特征值的情况下，该函数通过数组过滤方式将上述数据集合切分得到两个子集并返回。
    Args:
        dataMat 数据集
        feature 待切分的特征列
        value 特征列要比较的值
    Returns:
        mat0 小于等于 value 的数据集在左边
        mat1 大于 value 的数据集在右边
    Raises:
    """
    # # 测试案例
    # print 'dataSet[:, feature]=', dataSet[:, feature]
    # print 'nonzero(dataSet[:, feature] > value)[0]=', nonzero(dataSet[:, feature] > value)[0]
    # print 'nonzero(dataSet[:, feature] <= value)[0]=', nonzero(dataSet[:, feature] <= value)[0]

    # dataSet[:, feature] 取去每一行中，第1列的值(从0开始算)
    # nonzero(dataSet[:, feature] > value)  返回结果为true行的index下标
    mat0 = dataSet[nonzero(dataSet[:, feature] <= value)[0], :]
    mat1 = dataSet[nonzero(dataSet[:, feature] > value)[0], :]
    return mat0, mat1


# 1.用最佳方式切分数据集
# 2.生成相应的叶节点
def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
    """chooseBestSplit(用最佳方式切分数据集 和 生成相应的叶节点)

    Args:
        dataSet   加载的原始数据集
        leafType  建立叶子点的函数
        errType   误差计算函数(求总方差)
        ops       [容许误差下降值，切分的最少样本数]。
    Returns:
```

```
        bestIndex feature的index坐标
        bestValue 切分的最优值
    Raises:
    """

    # ops=(1,4)，非常重要，因为它决定了决策树划分停止的threshold值，被称为预剪枝（prepruning），其实也就是用于控制函数的停止时机。
    # 之所以这样说，是因为它防止决策树的过拟合，所以当误差的下降值小于tolS，或划分后的集合size小于tolN时，选择停止继续划分。
    # 最小误差下降值，划分后的误差减小小于这个差值，就不用继续划分
    tolS = ops[0]
    # 划分最小 size 小于，就不继续划分了
    tolN = ops[1]
    # 如果结果集(最后一列为1个变量)，就返回退出
    # .T 对数据集进行转置
    # .tolist()[0] 转化为数组并取第0列
    if len(set(dataSet[:, -1].T.tolist()[0])) == 1: # 如果集合size为1，不用继续划分。
        #  exit cond 1
        return None, leafType(dataSet)
    # 计算行列值
    m, n = shape(dataSet)
    # 无分类误差的总方差和
    # the choice of the best feature is driven by Reduction in RSS error from mean
    S = errType(dataSet)
    # inf 正无穷大
    bestS, bestIndex, bestValue = inf, 0, 0
    # 循环处理每一列对应的feature值
    for featIndex in range(n-1): # 对于每个特征
        # [0]表示这一列的[所有行]，不要[0]就是一个array[[所有行]]
        for splitVal in set(dataSet[:, featIndex].T.tolist()[0]):
            # 对该列进行分组，然后组内的成员的val值进行 二元切分
            mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
            # 判断二元切分的方式的元素数量是否符合预期
            if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN):
                continue
            newS = errType(mat0) + errType(mat1)
            # 如果二元切分，算出来的误差在可接受范围内，那么就记录切分点，并记录最小误差
            # 如果划分后误差小于 bestS，则说明找到了新的bestS
            if newS < bestS:
                bestIndex = featIndex
                bestValue = splitVal
                bestS = newS
    # 判断二元切分的方式的元素误差是否符合预期
    # if the decrease (S-bestS) is less than a threshold don't do the split
    if (S - bestS) < tolS:
        return None, leafType(dataSet)
    mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
    # 对整体的成员进行判断，是否符合预期
    # 如果集合的 size 小于 tolN
    if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN): # 当最佳划分后，集合过小，也不划分，产生叶节点
        return None, leafType(dataSet)
    return bestIndex, bestValue


# assume dataSet is NumPy Mat so we can array filtering
# 假设 dataSet 是 NumPy Mat 类型的，那么我们可以进行 array 过滤
def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1, 4)):
    """createTree(获取回归树)
        Description：递归函数：如果构建的是回归树，该模型是一个常数，如果是模型树，其模型师一个线性方程。
    Args:
        dataSet      加载的原始数据集
        leafType     建立叶子点的函数
        errType      误差计算函数
        ops=(1, 4)   [容许误差下降值，切分的最少样本数]
    Returns:
        retTree    决策树最后的结果
    """
    # 选择最好的切分方式： feature索引值，最优切分值
    # choose the best split
    feat, val = chooseBestSplit(dataSet, leafType, errType, ops)
    # if the splitting hit a stop condition return val
    # 如果 splitting 达到一个停止条件，那么返回 val
    if feat is None:
        return val
    retTree = {}
    retTree['spInd'] = feat
    retTree['spVal'] = val
    # 大于在右边，小于在左边，分为2个数据集
    lSet, rSet = binSplitDataSet(dataSet, feat, val)
    # 递归的进行调用，在左右子树中继续递归生成树
    retTree['left'] = createTree(lSet, leafType, errType, ops)
    retTree['right'] = createTree(rSet, leafType, errType, ops)
    return retTree
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/9.RegTrees/regTrees.py

> 测试算法：使用测试数据上的R^2值来分析模型的效果

> 使用算法：使用训练出的树做预测，预测结果还可以用来做很多事情

2、树剪枝

一棵树如果节点过多，表明该模型可能对数据进行了 “过拟合”。

通过降低决策树的复杂度来避免过拟合的过程称为 `剪枝（pruning）`。在函数 chooseBestSplit() 中提前终止条件，实际上是在进行一种所谓的 `预剪枝（prepruning）` 操作。另一个形式的剪枝需要使用测试集和训练集，称作 `后剪枝（postpruning）`。

2.1、预剪枝(prepruning)

顾名思义，预剪枝就是及早的停止树增长，在构造决策树的同时进行剪枝。

所有决策树的构建方法，都是在无法进一步降低熵的情况下才会停止创建分支的过程，为了避免过拟合，可以设定一个阈值，熵减小的数量小于这个阈值，即使还可以继续降低熵，也停止继续创建分支。但是这种方法实际中的效果并不好。

2.2、后剪枝(postpruning)

决策树构造完成后进行剪枝。剪枝的过程是对拥有同样父节点的一组节点进行检查，判断如果将其合并，熵的增加量是否小于某一阈值。如果确实小，则这一组节点可以合并一个节点，其中包含了所有可能的结果。合并也被称作 `塌陷处理` ，在回归树中一般采用取需要合并的所有子树的平均值。后剪枝是目前最普遍的做法。

后剪枝 prune() 的伪代码如下:

```
基于已有的树切分测试数据:
    如果存在任一子集是一棵树，则在该子集递归剪枝过程
    计算将当前两个叶节点合并后的误差
    计算不合并的误差
    如果合并会降低误差的话，就将叶节点合并
```

2.3、剪枝 代码

回归树剪枝函数

```
# 判断节点是否是一个字典
def isTree(obj):
    """
    Desc:
        测试输入变量是否是一棵树,即是否是一个字典
    Args:
        obj -- 输入变量
    Returns:
        返回布尔类型的结果。如果 obj 是一个字典，返回true，否则返回 false
    """
    return (type(obj).__name__ == 'dict')


# 计算左右枝丫的均值
def getMean(tree):
    """
    Desc:
        从上往下遍历树直到叶节点为止，如果找到两个叶节点则计算它们的平均值。
        对 tree 进行塌陷处理，即返回树平均值。
    Args:
        tree -- 输入的树
    Returns:
        返回 tree 节点的平均值
    """
    if isTree(tree['right']):
        tree['right'] = getMean(tree['right'])
    if isTree(tree['left']):
        tree['left'] = getMean(tree['left'])
    return (tree['left']+tree['right'])/2.0


# 检查是否适合合并分枝
def prune(tree, testData):
    """
    Desc:
        从上而下找到叶节点，用测试数据集来判断将这些叶节点合并是否能降低测试误差
    Args:
        tree -- 待剪枝的树
        testData -- 剪枝所需要的测试数据 testData
    Returns:
        tree -- 剪枝完成的树
    """
    # 判断是否测试数据集没有数据，如果没有，就直接返回tree本身的均值
```

```
    if shape(testData)[0] == 0:
        return getMean(tree)

    # 判断分枝是否是dict字典，如果是就将测试数据集进行切分
    if (isTree(tree['right']) or isTree(tree['left'])):
        lSet, rSet = binSplitDataSet(testData, tree['spInd'], tree['spVal'])
    # 如果是左边分枝是字典，就传入左边的数据集和左边的分枝，进行递归
    if isTree(tree['left']):
        tree['left'] = prune(tree['left'], lSet)
    # 如果是右边分枝是字典，就传入左边的数据集和左边的分枝，进行递归
    if isTree(tree['right']):
        tree['right'] = prune(tree['right'], rSet)

    # 上面的一系列操作本质上就是将测试数据集按照训练完成的树拆分好，对应的值放到对应的节点

    # 如果左右两边同时都不是dict字典，也就是左右两边都是叶节点，而不是子树了，那么分割测试数据集。
    # 1. 如果正确
    #   * 那么计算一下总方差 和 该结果集的本身不分枝的总方差比较
    #   * 如果 合并的总方差 < 不合并的总方差，那么就进行合并
    # 注意返回的结果： 如果可以合并，原来的dict就变为了 数值
    if not isTree(tree['left']) and not isTree(tree['right']):
        lSet, rSet = binSplitDataSet(testData, tree['spInd'], tree['spVal'])
        # power(x, y)表示x的y次方
        errorNoMerge = sum(power(lSet[:, -1] - tree['left'], 2)) + sum(power(rSet[:, -1] - tree['right'], 2))
        treeMean = (tree['left'] + tree['right'])/2.0
        errorMerge = sum(power(testData[:, -1] - treeMean, 2))
        # 如果 合并的总方差 < 不合并的总方差，那么就进行合并
        if errorMerge < errorNoMerge:
            print "merging"
            return treeMean
        else:
            return tree
    else:
        return tree
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/9.RegTrees/regTrees.py

3、模型树

3.1、模型树 简介

用树来对数据建模，除了把叶节点简单地设定为常数值之外，还有一种方法是把叶节点设定为分段线性函数，这里所谓的 `分段线性（piecewise linear）` 是指模型由多个线性片段组成。

我们看一下图 9-4 中的数据，如果使用两条直线拟合是否比使用一组常数来建模好呢？答案显而易见。可以设计两条分别从 0.0~~0.3、从 0.3~~1.0 的直线，于是就可以得到两个线性模型。因为数据集里的一部分数据（0.0~~0.3）以某个线性模型建模，而另一部分数据（0.3~~1.0）则以另一个线性模型建模，因此我们说采用了所谓的分段线性模型。

决策树相比于其他机器学习算法的优势之一在于结果更易理解。很显然，两条直线比很多节点组成一棵大树更容易解释。模型树的可解释性是它优于回归树的特点之一。另外，模型树也具有更高的预测准确度。

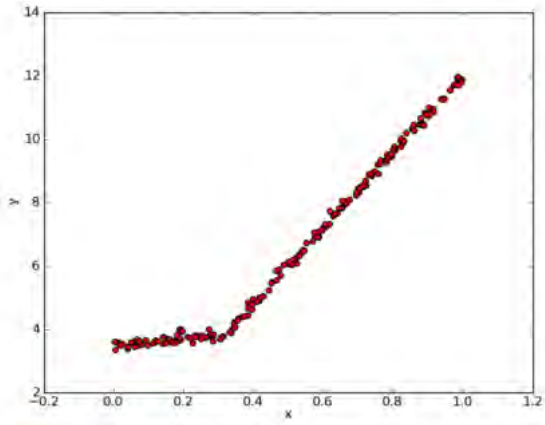


Figure 9.4 **Piecewise linear data to test model tree-creation functions**

将之前的回归树的代码稍作修改，就可以在叶节点生成线性模型而不是常数值。下面将利用树生成算法对数据进行划分，且每份切分数据都能很容易被线性模型所表示。这个算法的关键在于误差的计算。

那么为了找到最佳切分，应该怎样计算误差呢？前面用于回归树的误差计算方法这里不能再用。稍加变化，对于给定的数据集，应该先用模型来对它进行拟合，然后计算真实的目标值与模型预测值间的差值。最后将这些差值的平方求和就得到了所需的误差。

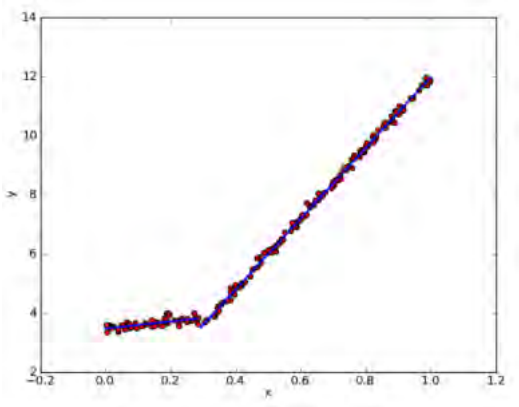
3.2、模型树 代码

模型树的叶节点生成函数

```
# 得到模型的ws系数：f(x) = x0 + x1*featrue1+ x3*featrue2 ...
# create linear model and return coeficients
def modelLeaf(dataSet):
    """
    Desc:
        当数据不再需要切分的时候，生成叶节点的模型。
    Args:
        dataSet -- 输入数据集
    Returns:
        调用 linearSolve 函数，返回得到的 回归系数ws
    """
    ws, X, Y = linearSolve(dataSet)
    return ws


# 计算线性模型的误差值
def modelErr(dataSet):
    """
    Desc:
        在给定数据集上计算误差。
    Args:
        dataSet -- 输入数据集
    Returns:
        调用 linearSolve 函数，返回 yHat 和 Y 之间的平方误差。
    """
    ws, X, Y = linearSolve(dataSet)
    yHat = X * ws
    # print corrcoef(yHat, Y, rowvar=0)
    return sum(power(Y - yHat, 2))


 # helper function used in two places
def linearSolve(dataSet):
    """
    Desc:
        将数据集格式化成目标变量Y和自变量X，执行简单的线性回归，得到ws
    Args:
        dataSet -- 输入数据
    Returns:
        ws -- 执行线性回归的回归系数
        X -- 格式化自变量X
        Y -- 格式化目标变量Y
    """
    m, n = shape(dataSet)
    # 产生一个关于1的矩阵
    X = mat(ones((m, n)))
    Y = mat(ones((m, 1)))
    # X的0列为1，常数项，用于计算平衡误差
    X[:, 1: n] = dataSet[:, 0: n-1]
    Y = dataSet[:, -1]

    # 转置矩阵*矩阵
    xTx = X.T * X
    # 如果矩阵的逆不存在，会造成程序异常
    if linalg.det(xTx) == 0.0:
        raise NameError('This matrix is singular, cannot do inverse,\ntry increasing the second value of ops')
    # 最小二乘法求最优解:  w0*1+w1*x1=y
    ws = xTx.I * (X.T * Y)
    return ws, X, Y
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/9.RegTrees/regTrees.py

3.3、模型树 运行结果



4、树回归 项目案例

4.1、项目案例1: 树回归与标准回归的比较

4.1.1、项目概述

前面介绍了模型树、回归树和一般的回归方法，下面测试一下哪个模型最好。

这些模型将在某个数据上进行测试，该数据涉及人的智力水平和自行车的速度的关系。当然，数据是假的。

4.1.2、开发流程

```
收集数据：采用任意方法收集数据
准备数据：需要数值型数据，标称型数据应该映射成二值型数据
分析数据：绘出数据的二维可视化显示结果，以字典方式生成树
训练算法：模型树的构建
测试算法：使用测试数据上的R^2值来分析模型的效果
使用算法：使用训练出的树做预测，预测结果还可以用来做很多事情
```

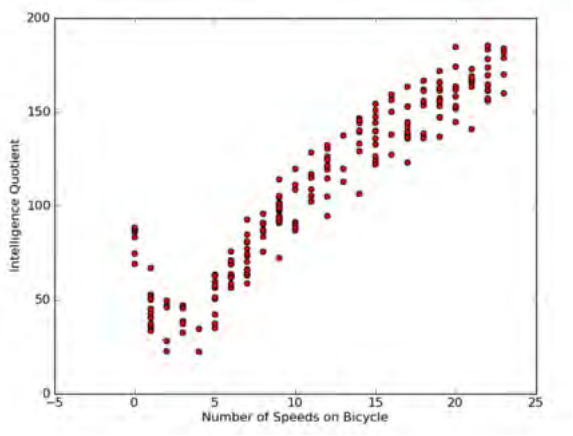
> 收集数据: 采用任意方法收集数据

> 准备数据：需要数值型数据，标称型数据应该映射成二值型数据

数据存储格式:

```
3.000000        46.852122
23.000000       178.676107
0.000000        86.154024
6.000000        68.707614
15.000000       139.737693
```

> 分析数据：绘出数据的二维可视化显示结果，以字典方式生成树



> 训练算法：模型树的构建

用树回归进行预测的代码

```
# 回归树测试案例
# 为了和 modelTreeEval() 保持一致，保留两个输入参数
def regTreeEval(model, inDat):
    """
    Desc:
        对 回归树 进行预测
    Args:
        model -- 指定模型，可选值为 回归树模型 或者 模型树模型，这里为回归树
        inDat -- 输入的测试数据
    Returns:
        float(model) -- 将输入的模型数据转换为 浮点数 返回
    """
    return float(model)


# 模型树测试案例
# 对输入数据进行格式化处理，在原数据矩阵上增加第0列，元素的值都是1，
# 也就是增加偏移值，和我们之前的简单线性回归是一个套路，增加一个偏移量
def modelTreeEval(model, inDat):
    """
    Desc:
        对 模型树 进行预测
    Args:
        model -- 输入模型，可选值为 回归树模型 或者 模型树模型，这里为模型树模型
        inDat -- 输入的测试数据
```

```
    Returns:
        float(X * model) -- 将测试数据乘以 回归系数 得到一个预测值 ，转化为 浮点数 返回
    """
    n = shape(inDat)[1]
    X = mat(ones((1, n+1)))
    X[:, 1: n+1] = inDat
    # print X, model
    return float(X * model)


# 计算预测的结果
# 在给定树结构的情况下，对于单个数据点，该函数会给出一个预测值。
# modelEval是对叶节点进行预测的函数引用，指定树的类型，以便在叶节点上调用合适的模型。
# 此函数自顶向下遍历整棵树，直到命中叶节点为止，一旦到达叶节点，它就会在输入数据上
# 调用modelEval()函数，该函数的默认值为regTreeEval()
def treeForeCast(tree, inData, modelEval=regTreeEval):
    """
    Desc:
        对特定模型的树进行预测，可以是 回归树 也可以是 模型树
    Args:
        tree -- 已经训练好的树的模型
        inData -- 输入的测试数据
        modelEval -- 预测的树的模型类型，可选值为 regTreeEval（回归树） 或 modelTreeEval（模型树），默认为回归树
    Returns:
        返回预测值
    """
    if not isTree(tree):
        return modelEval(tree, inData)
    if inData[tree['spInd']] <= tree['spVal']:
        if isTree(tree['left']):
            return treeForeCast(tree['left'], inData, modelEval)
        else:
            return modelEval(tree['left'], inData)
    else:
        if isTree(tree['right']):
            return treeForeCast(tree['right'], inData, modelEval)
        else:
            return modelEval(tree['right'], inData)


# 预测结果
def createForeCast(tree, testData, modelEval=regTreeEval):
    """
    Desc:
        调用 treeForeCast ，对特定模型的树进行预测，可以是 回归树 也可以是 模型树
    Args:
        tree -- 已经训练好的树的模型
        inData -- 输入的测试数据
        modelEval -- 预测的树的模型类型，可选值为 regTreeEval（回归树） 或 modelTreeEval（模型树），默认为回归树
    Returns:
        返回预测值矩阵
    """
    m = len(testData)
    yHat = mat(zeros((m, 1)))
    # print yHat
    for i in range(m):
        yHat[i, 0] = treeForeCast(tree, mat(testData[i]), modelEval)
        # print "yHat==>", yHat[i, 0]
    return yHat
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/9.RegTrees/regTrees.py

> 测试算法：使用测试数据上的R^2值来分析模型的效果

R^2 判定系数就是拟合优度判定系数，它体现了回归模型中自变量的变异在因变量的变异中所占的比例。如 R^2=0.99999 表示在因变量 y 的变异中有 99.999% 是由于变量 x 引起。当 R^2=1 时表示，所有观测点都落在拟合的直线或曲线上；当 R^2=0 时，表示自变量与因变量不存在直线或曲线关系。

所以我们看出， R^2 的值越接近 1.0 越好。

> 使用算法：使用训练出的树做预测，预测结果还可以用来做很多事情

5、附加 Python 中 GUI 的使用

5.1、使用 Python 的 Tkinter 库创建 GUI

如果能让用户不需要任何指令就可以按照他们自己的方式来分析数据，就不需要对数据做出过多解释。其中一个能同时支持数据呈现和用户交互的方式就是构建一个图形用户界面(GUI，Graphical User Interface)，如图9-7所示。

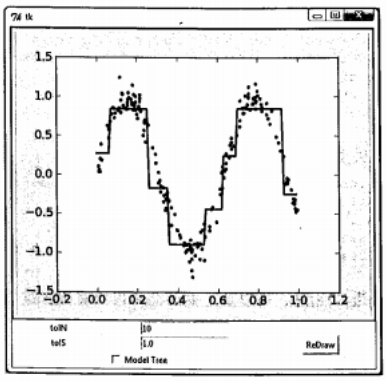


图9-7 默认的treeExplore图形用户界面，该界面同时显示了输入数据和一个回归树模型，其中的参数tolN=10，tolS=1.0

5.2、用 Tkinter 创建 GUI

Python 有很多 GUI 框架，其中一个易于使用的 Tkinter，是随 Python 的标准版编译版本发布的。Tkinter 可以在 Windows、Mac OS和大多数的 Linux 平台上使用。

5.3、集成 Matplotlib 和 Tkinter

MatPlotlib 的构建程序包含一个前端，也就是面向用户的一些代码，如 plot() 和 scatter() 方法等。事实上，它同时创建了一个后端，用于实现绘图和不同应用之间接口。

通过改变后端可以将图像绘制在PNG、PDF、SVG等格式的文件上。下面将设置后端为 TkAgg (Agg 是一个 C++ 的库，可以从图像创建光栅图)。TkAgg可以在所选GUI框架上调用Agg，把 Agg 呈现在画布上。我们可以在Tk的GUI上放置一个画布，并用 .grid()来调整布局。

5.4、用treeExplore 的GUI构建的模型树示例图

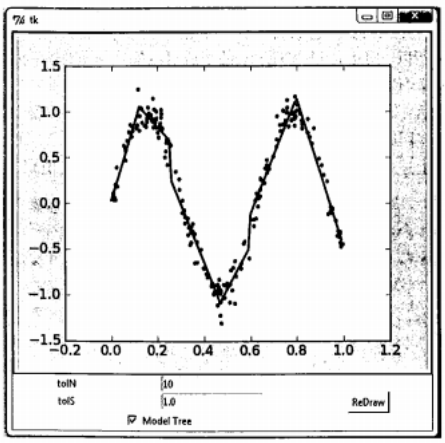


图9-9 用treeExplore的GUI构建的模型树，该图使用了与图9-7相同的数据和参数。与回归树相比，模型树取得了更好的预测效果

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/9.RegTrees/treeExplore.py

6、树回归 小结

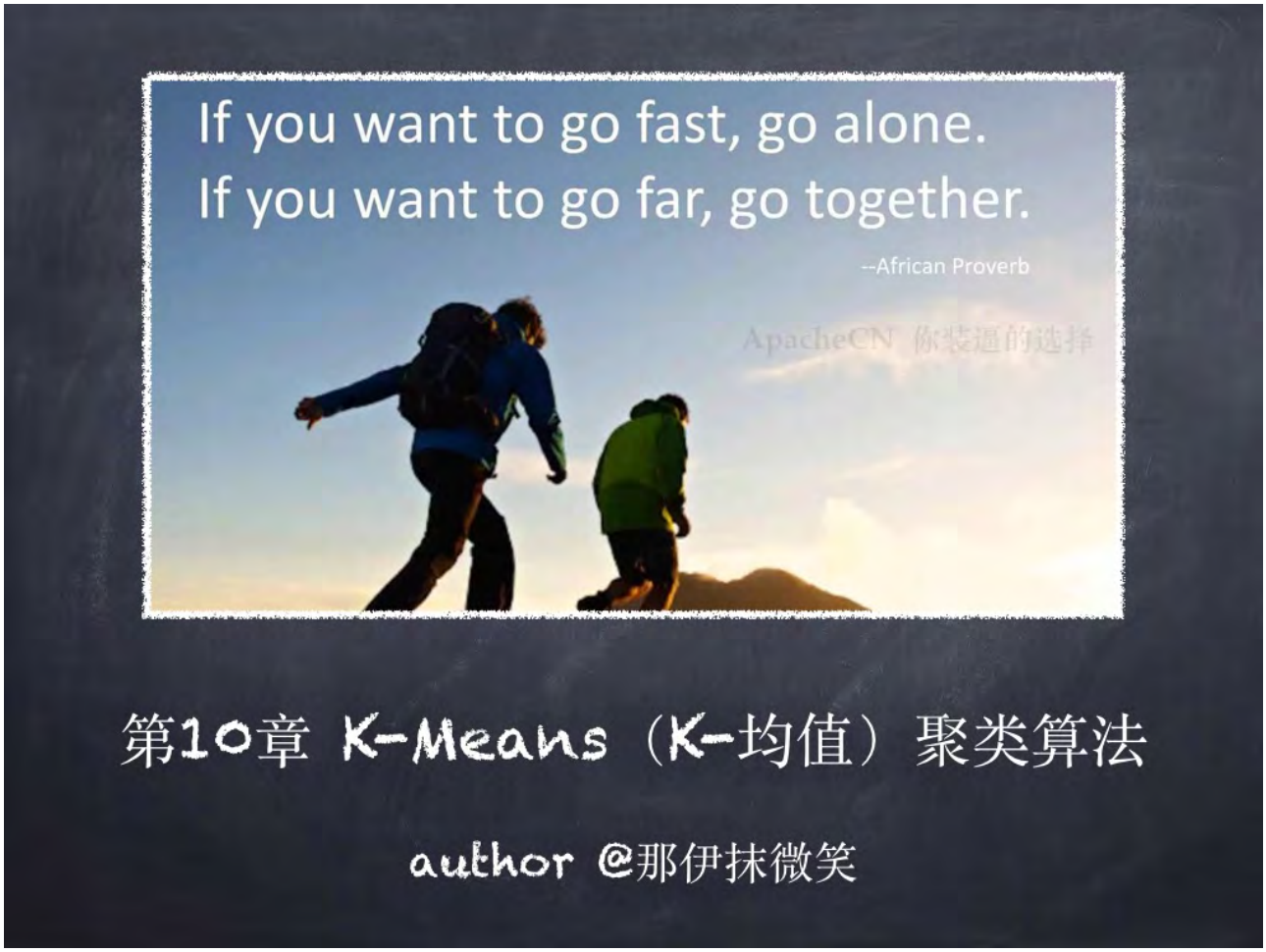
数据集中经常包含一些复杂的相关关系，使得输入数据和目标变量之间呈现非线性关系。对这些复杂的关系建模，一种可行的方式是使用树来对预测值分段，包括分段常数或分段直线。一般采用树结构来对这种数据建模。相应地，若叶节点使用的模型是分段常数则称为回归树，若叶节点使用的模型师线性回归方程则称为模型树。

CART 算法可以用于构建二元树并处理离散型或连续型数据的切分。若使用不同的误差准则，就可以通过CART 算法构建模型树和回归树。该算法构建出的树会倾向于对数据过拟合。一棵过拟合的树常常十分复杂，剪枝技术的出现就是为了解决这个问题。两种剪枝方法分别是预剪枝（在树的构建过程中就进行剪枝）和后剪枝（当树构建完毕再进行剪枝），预剪枝更有效但需要用户定义一些参数。

Tkinter 是 Python 的一个 GUI 工具包。虽然并不是唯一的包，但它最常用。利用 Tkinter ，我们可以轻轻松松绘制各种部件并安排它们的位置。另外，可以为 Tkinter 构造一个特殊的部件来显示 Matplotlib 绘出的图。所以，Matplotlib 和 Tkinter 的集成可以构建出更强大的 GUI ，用户可以以更自然的方式来探索机器学习算法的奥妙。

- 作者：片刻 小瑶
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第 10 章 K-Means（K-均值）聚类算法



K-Means 算法

聚类是一种无监督的学习, 它将相似的对象归到一个簇中, 将不相似对象归到不同簇中.
相似这一概念取决于所选择的相似度计算方法.
K-Means 是发现给定数据集的 K 个簇的聚类算法, 之所以称之为 `K-均值` 是因为它可以发现 K 个不同的簇, 且每个簇的中心采用簇中所含值的均值计算而成.
簇个数 K 是用户指定的, 每一个簇通过其质心（centroid）, 即簇中所有点的中心来描述.
聚类与分类算法的最大区别在于, 分类的目标类别已知, 而聚类的目标类别是未知的.

```
优点：容易实现
缺点:可能收敛到局部最小值，在大规模数据集上收敛较慢
使用数据类型 ：数值型数据
```

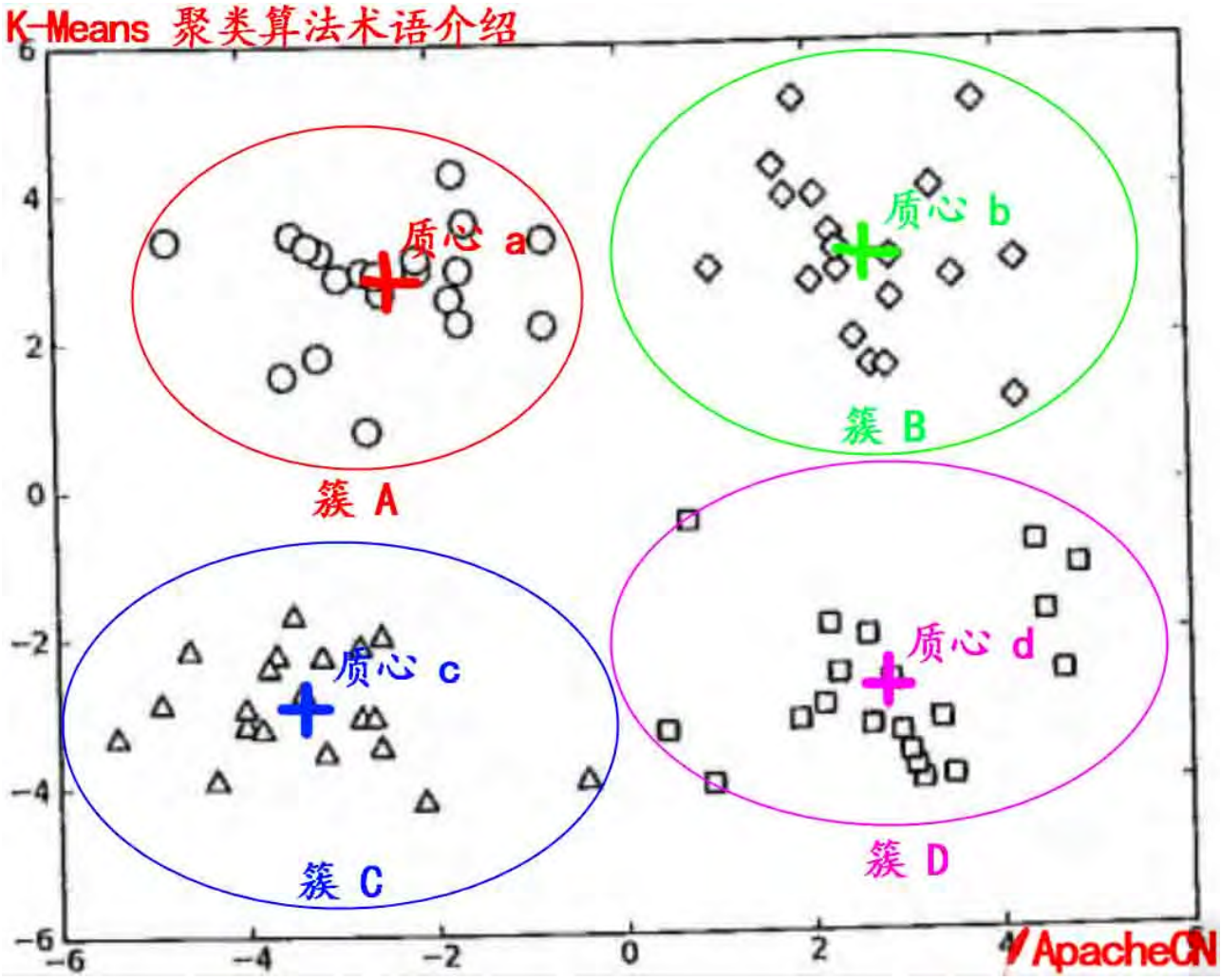
K-Means 场景

主要用来聚类, 但是类别是未知的.
例如: 对地图上的点进行聚类.

K-Means 术语

- 簇: 所有数据点点集合，簇中的对象是相似的。
- 质心: 簇中所有点的中心（计算所有点的均值而来）.
- SSE: Sum of Sqared Error（平方误差和）, SSE 值越小，表示越接近它们的质心. 由于对误差取了平方，因此更加注重那么远离中心的点.

有关 `簇` 和 `质心` 术语更形象的介绍, 请参考下图:



K-Means 工作流程

1. 首先, 随机确定 K 个初始点作为质心（不是数据中的点）.
2. 然后将数据集中的每个点分配到一个簇中, 具体来讲, 就是为每个点找到距其最近的质心, 并将其分配该质心所对应的簇. 这一步完成之后, 每个簇的质心更新为该簇说有点的平均值.

上述过程的 伪代码 如下:

- 创建 k 个点作为起始质心（通常是随机选择）
- 当任意一个点的簇分配结果发生改变时
  - 对数据集中的每个数据点
    - 对每个质心
      - 计算质心与数据点之间的距离
    - 将数据点分配到距其最近的簇
  - 对每一个簇, 计算簇中所有点的均值并将均值作为质心

K-Means 开发流程

```
收集数据：使用任意方法
准备数据：需要数值型数据类计算距离，也可以将标称型数据映射为二值型数据再用于距离计算
分析数据：使用任意方法
训练算法：此步骤不适用于 K-Means 算法
测试算法：应用聚类算法、观察结果.可以使用量化的误差指标如误差平方和（后面会介绍）来评价算法的结果.
使用算法：可以用于所希望的任何应用.通常情况下，簇质心可以代表整个簇的数据来做出决策.
```

K-Means 聚类算法函数

从文件加载数据集

```
# 从文本中构建矩阵，加载文本文件，然后处理
def loadDataSet(fileName):      # 通用函数，用来解析以 tab 键分隔的 floats（浮点数），例如: 1.658985  4.285136
    dataMat = []
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = map(float,curLine)    # 映射所有的元素为 float（浮点数）类型
        dataMat.append(fltLine)
    return dataMat
```

计算两个向量的欧氏距离

```
# 计算两个向量的欧式距离（可根据场景选择）
def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2))) # la.norm(vecA-vecB)
```

构建一个包含 K 个随机质心的集合

```
# 为给定数据集构建一个包含 k 个随机质心的集合。随机质心必须要在整个数据集的边界之内，这可以通过找到数据集每一维的最小和最大值来完成。然后生成 0~1
def randCent(dataSet, k):
    n = shape(dataSet)[1] # 列的数量
    centroids = mat(zeros((k,n))) # 创建k个质心矩阵
    for j in range(n): # 创建随机簇质心，并且在每一维的边界内
        minJ = min(dataSet[:,j])    # 最小值
        rangeJ = float(max(dataSet[:,j]) - minJ)    # 范围 = 最大值 - 最小值
        centroids[:,j] = mat(minJ + rangeJ * random.rand(k,1))    # 随机生成
    return centroids
```

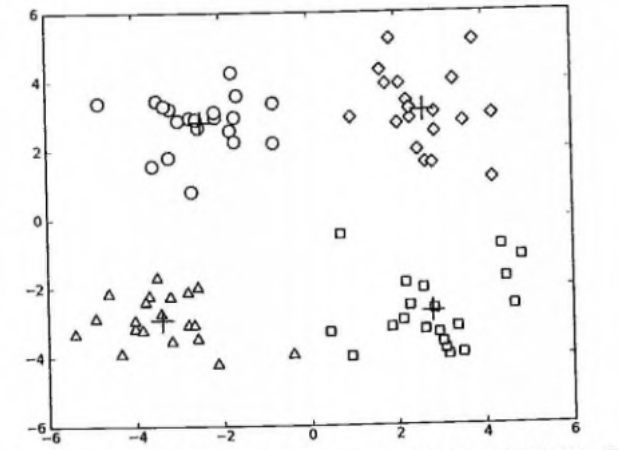
K-Means 聚类算法

```
# k-means 聚类算法
# 该算法会创建k个质心，然后将每个点分配到最近的质心，再重新计算质心。
# 这个过程重复数次，直到数据点的簇分配结果不再改变位置。
# 运行结果（多次运行结果可能会不一样，可以试试，原因为随机质心的影响，但总的结果是对的， 因为数据足够相似，也可能会陷入局部最小值）
def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = shape(dataSet)[0]    # 行数
    clusterAssment = mat(zeros((m, 2)))    # 创建一个与 dataSet 行数一样，但是有两列的矩阵，用来保存簇分配结果
    centroids = createCent(dataSet, k)    # 创建质心，随机k个质心
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):    # 循环每一个数据点并分配到最近的质心中去
            minDist = inf; minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j,:],dataSet[i,:])    # 计算数据点到质心的距离
                if distJI < minDist:    # 如果距离比 minDist（最小距离）还小，更新 minDist（最小距离）和最小质心的 index（索引）
                    minDist = distJI; minIndex = j
            if clusterAssment[i, 0] != minIndex:    # 簇分配结果改变
                clusterChanged = True    # 簇改变
                clusterAssment[i, :] = minIndex,minDist**2    # 更新簇分配结果为最小质心的 index（索引），minDist（最小距离）的平方
        print centroids
        for cent in range(k): # 更新质心
            ptsInClust = dataSet[nonzero(clusterAssment[:, 0].A==cent)[0]] # 获取该簇中的所有点
            centroids[cent,:] = mean(ptsInClust, axis=0) # 将质心修改为簇中所有点的平均值，mean 就是求平均值的
    return centroids, clusterAssment
```

测试函数

1. 测试一下以上的基础函数是否可以如预期运行, 请看: https://github.com/apachecn/MachineLearning/blob/master/src/python/10.kmeans/kMeans.py
2. 测试一下 kMeans 函数是否可以如预期运行, 请看: https://github.com/apachecn/MachineLearning/blob/master/src/python/10.kmeans/kMeans.py

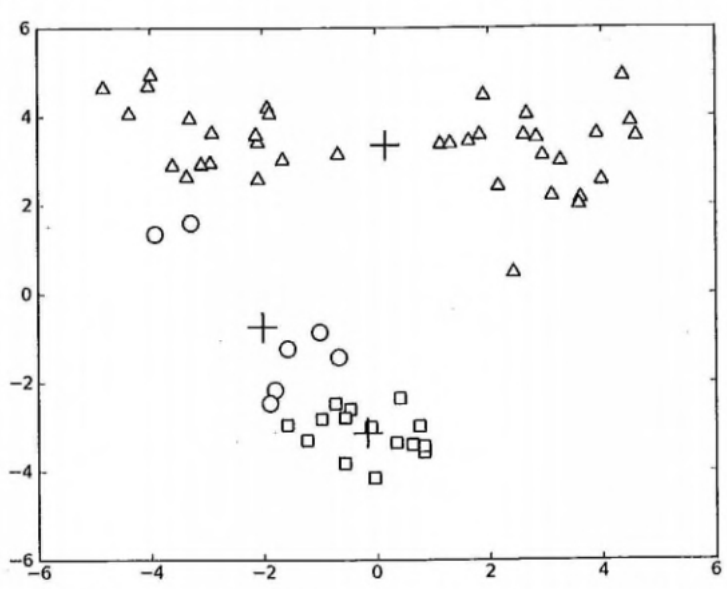
参考运行结果如下:



> 在 kMeans 的函数测试中，可能偶尔会陷入局部最小值（局部最优的结果，但不是全局最优的结果）.

K-Means 聚类算法的缺陷

在 kMeans 的函数测试中，可能偶尔会陷入局部最小值（局部最优的结果，但不是全局最优的结果）.
局部最小值的的情况如下:



所以为了克服 KMeans 算法收敛于局部最小值的问题，有更厉害的大佬提出了另一个称之为二分K-均值（bisecting K-Means）的算法.

二分 K-Means 聚类算法

该算法首先将所有点作为一个簇，然后将该簇一分为二。
之后选择其中一个簇继续进行划分，选择哪一个簇进行划分取决于对其划分时候可以最大程度降低 SSE（平方和误差）的值。
上述基于 SSE 的划分过程不断重复，直到得到用户指定的簇数目为止。

二分 K-Means 聚类算法伪代码

- 将所有点看成一个簇
- 当簇数目小雨 k 时
- 对于每一个簇
  - 计算总误差
  - 在给定的簇上面进行 KMeans 聚类（k=2）
  - 计算将该簇一分为二之后的总误差
- 选择使得误差最小的那个簇进行划分操作

另一种做法是选择 SSE 最大的簇进行划分，直到簇数目达到用户指定的数目位置。 接下来主要介绍该做法。

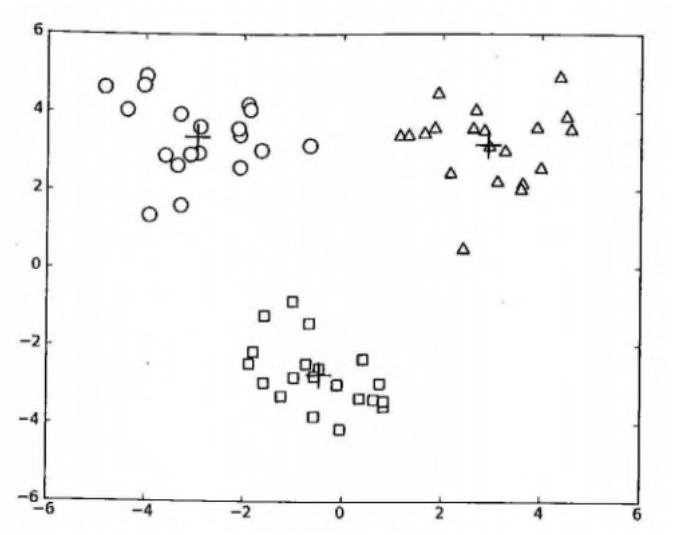
二分 K-Means 聚类算法代码

```
# 二分 KMeans 聚类算法，基于 kMeans 基础之上的优化，以避免陷入局部最小值
def biKMeans(dataSet, k, distMeas=distEclud):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m,2))) # 保存每个数据点的簇分配结果和平方误差
    centroid0 = mean(dataSet, axis=0).tolist()[0] # 质心初始化为所有数据点的均值
    centList =[centroid0] # 初始化只有 1 个质心的 list
    for j in range(m): # 计算所有数据点到初始质心的距离平方误差
        clusterAssment[j,1] = distMeas(mat(centroid0), dataSet[j,:])**2
    while (len(centList) < k): # 当质心数量小于 k 时
        lowestSSE = inf
        for i in range(len(centList)): # 对每一个质心
            ptsInCurrCluster = dataSet[nonzero(clusterAssment[:,0].A==i)[0],:] # 获取当前簇 i 下的所有数据点
            centroidMat, splitClustAss = kMeans(ptsInCurrCluster, 2, distMeas) # 将当前簇 i 进行二分 kMeans 处理
            sseSplit = sum(splitClustAss[:,1]) # 将二分 kMeans 结果中的平方和的距离进行求和
            sseNotSplit = sum(clusterAssment[nonzero(clusterAssment[:,0].A!=i)[0],1]) # 将未参与二分 kMeans 分配结果中的平方和的距离进行
            print "sseSplit, and notSplit: ",sseSplit,sseNotSplit
            if (sseSplit + sseNotSplit) < lowestSSE: # 总的（未拆分和已拆分）误差和越小，越相似，效果越优化，划分的结果更好（注意：这里的理解
                bestCentToSplit = i
                bestNewCents = centroidMat
                bestClustAss = splitClustAss.copy()
                lowestSSE = sseSplit + sseNotSplit
        # 找出最好的簇分配结果
        bestClustAss[nonzero(bestClustAss[:,0].A == 1)[0],0] = len(centList) # 调用二分 kMeans 的结果，默认簇是 0,1. 当然也可以改成其它的
        bestClustAss[nonzero(bestClustAss[:,0].A == 0)[0],0] = bestCentToSplit # 更新为最佳质心
        print 'the bestCentToSplit is: ',bestCentToSplit
        print 'the len of bestClustAss is: ', len(bestClustAss)
        # 更新质心列表
        centList[bestCentToSplit] = bestNewCents[0,:].tolist()[0] # 更新原质心 list 中的第 i 个质心为使用二分 kMeans 后 bestNewCents 的第
        centList.append(bestNewCents[1,:].tolist()[0]) # 添加 bestNewCents 的第二个质心
        clusterAssment[nonzero(clusterAssment[:,0].A == bestCentToSplit)[0],:]= bestClustAss # 重新分配最好簇下的数据（质心）以及SSE
    return mat(centList), clusterAssment
```

测试二分 KMeans 聚类算法

- 测试一下二分 KMeans 聚类算法，请看: https://github.com/apachecn/MachineLearning/blob/master/src/python/10.kmeans/kMeans.py

上述函数可以运行多次，聚类会收敛到全局最小值，而原始的 kMeans() 函数偶尔会陷入局部最小值。
运行参考结果如下:



- 作者：那伊抹微笑
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第 11 章 使用 Apriori 算法进行关联分析

关联分析

关联分析是一种在大规模数据集中寻找有趣关系的任务。 这些关系可以有两种形式:

- 频繁项集（frequent item sets）: 经常出现在一块的物品的集合。
- 关联规则（associational rules）: 暗示两种物品之间可能存在很强的关系。

相关术语

- 关联分析（关联规则学习): 从大规模数据集中寻找物品间的隐含关系被称作 `关联分析(associati analysis)` 或者 `关联规则学习（association rule learning）` 。 下面是用一个 `杂货店` 例子来说明这两个概念，如下图所示:

交易号码	商品
0	豆奶，莴苣
1	莴苣，尿布，葡萄酒，甜菜
2	豆奶，尿布，葡萄酒，橙汁
3	莴苣，豆奶，尿布，葡萄酒
4	莴苣，豆奶，尿布，橙汁

图11-1　一个来自Hole Foods天然食品店的简单交易清单

- 频繁项集: {葡萄酒, 尿布, 豆奶} 就是一个频繁项集的例子。
- 关联规则: 尿布 -> 葡萄酒 就是一个关联规则。这意味着如果顾客买了尿布，那么他很可能会买葡萄酒。

那么 `频繁` 的定义是什么呢？怎么样才算频繁呢？ 度量它们的方法有很多种，这里我们来简单的介绍下支持度和可信度。

- 支持度: 数据集中包含该项集的记录所占的比例。例如上图中，{豆奶} 的支持度为 4/5。{豆奶, 尿布} 的支持度为 3/5。
- 可信度: 针对一条诸如 {尿布} -> {葡萄酒} 这样具体的关联规则来定义的。这条规则的 `可信度` 被定义为 `支持度({尿布，葡萄酒})/支持度({尿布})`，从图中可以看出 支持度({尿布, 葡萄酒}) = 3/5，支持度({尿布}) = 4/5，所以 {尿布} -> {葡萄酒} 的可信度 = 3/5 / 4/5 = 3/4 = 0.75。

`支持度` 和 `可信度` 是用来量化 `关联分析` 是否成功的一个方法。 假设想找到支持度大于 0.8 的所有项集，应该如何去做呢？ 一个办法是生成一个物品所有可能组合的清单，然后对每一种组合统计它出现的频繁程度，但是当物品成千上万时，上述做法就非常非常慢了。 我们需要详细分析下这种情况并讨论下 Apriori 原理，该原理会减少关联规则学习时所需的计算量。

Apriori 原理

假设我们一共有 4 个商品: 商品0, 商品1, 商品2, 商品3。 所有可能的情况如下:

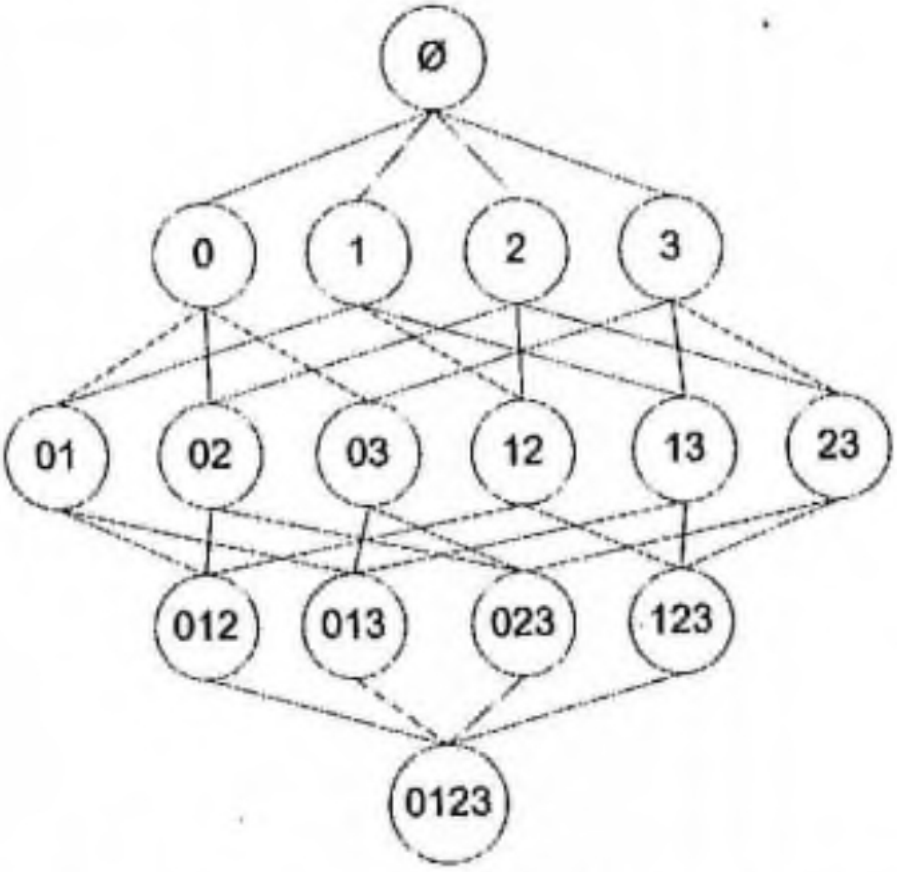
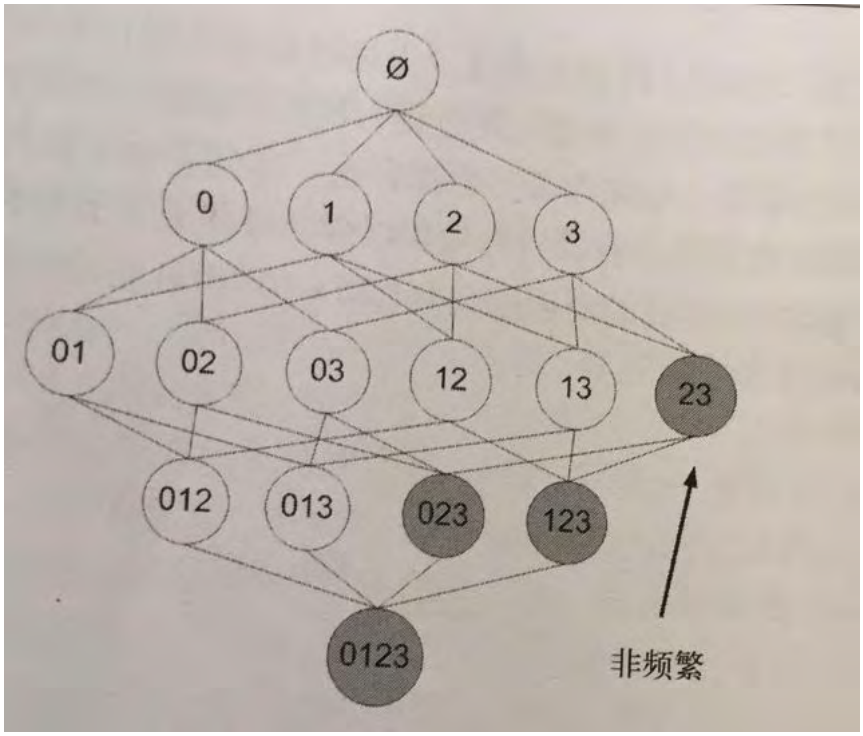


图11-2　集合{0，1，2，3}中所有可能的项集组合

如果我们计算所有组合的支持度，也需要计算 15 次。即 2^N - 1 = 2^4 - 1 = 15。
随着物品的增加，计算的次数呈指数的形式增长 ...
为了降低计算次数和时间，研究人员发现了一种所谓的 Apriori 原理，即某个项集是频繁的，那么它的所有子集也是频繁的。 例如，如果 {0, 1} 是频繁的，

那么 {0}, {1} 也是频繁的。 该原理直观上没有什么帮助，但是如果反过来看就有用了，也就是说如果一个项集是 非频繁项集，那么它的所有超集也是非频繁项集，如下图所示:



在图中我们可以看到，已知灰色部分 {2,3} 是 非频繁项集，那么利用上面的知识，我们就可以知道 {0,2,3} {1,2,3} {0,1,2,3} 都是 非频繁的。 也就是说，计算出 {2,3} 的支持度，知道它是 非频繁 的之后，就不需要再计算 {0,2,3} {1,2,3} {0,1,2,3} 的支持度，因为我们知道这些集合不会满足我们的要求。 使用该原理就可以避免项集数目的指数增长，从而在合理的时间内计算出频繁项集。

Apriori 算法优缺点

```
* 优点：易编码实现
* 缺点：在大数据集上可能较慢
* 适用数据类型：数值型 或者 标称型数据。
```

Apriori 算法流程步骤：

```
* 收集数据：使用任意方法。
* 准备数据：任何数据类型都可以，因为我们只保存集合。
* 分析数据：使用任意方法。
* 训练数据：使用Apiori算法来找到频繁项集。
* 测试算法：不需要测试过程。
* 使用算法：用语发现频繁项集以及物品之间的关联规则。
```

Apriori 算法的使用

前面提到，关联分析的目标包括两项: 发现 频繁项集 和发现 关联规则。 首先需要找到 频繁项集，然后才能发现 关联规则。
Apriori 算法是发现 频繁项集 的一种方法。 Apriori 算法的两个输入参数分别是最小支持度和数据集。 该算法首先会生成所有单个物品的项集列表。 接着扫描交易记录来查看哪些项集满足最小支持度要求，那些不满足最小支持度要求的集合会被去掉。 燃尽后对生下来的集合进行组合以声场包含两个元素的项集。 接下来再重新扫描交易记录，去掉不满足最小支持度的项集。 该过程重复进行直到所有项集被去掉。

生成候选项集

下面会创建一个用于构建初始集合的函数，也会创建一个通过扫描数据集以寻找交易记录子集的函数， 数据扫描的伪代码如下:

- 对数据集中的每条交易记录 tran
- 对每个候选项集 can
  - 检查一下 can 是否是 tran 的子集: 如果是则增加 can 的计数值
- 对每个候选项集
  - 如果其支持度不低于最小值，则保留该项集
  - 返回所有频繁项集列表 以下是一些辅助函数。

加载数据集

```
# 加载数据集
def loadDataSet():
    return [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]
```

创建集合 C1。即对 dataSet 进行去重，排序，放入 list 中，然后转换所有的元素为 frozenset

```
# 创建集合 C1。即对 dataSet 进行去重，排序，放入 list 中，然后转换所有的元素为 frozenset
def createC1(dataSet):
    """createC1（创建集合 C1）

    Args:
        dataSet 原始数据集
    Returns:
        frozenset 返回一个 frozenset 格式的 list
    """

    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                # 遍历所有的元素，如果不在 C1 出现过，那么就 append
                C1.append([item])
    # 对数组进行 `从小到大` 的排序
    print 'sort 前=', C1
    C1.sort()
    # frozenset 表示冻结的 set 集合，元素无改变；可以把它当字典的 key 来使用
    print 'sort 后=', C1
    print 'frozenset=', map(frozenset, C1)
    return map(frozenset, C1)
```

计算候选数据集 CK 在数据集 D 中的支持度，并返回支持度大于最小支持度（minSupport ）的数据

```
# 计算候选数据集 CK 在数据集 D 中的支持度，并返回支持度大于最小支持度（minSupport）的数据
def scanD(D, Ck, minSupport):
    """scanD（计算候选数据集 CK 在数据集 D 中的支持度，并返回支持度大于最小支持度 minSupport 的数据）

    Args:
        D 数据集
        Ck 候选项集列表
        minSupport 最小支持度
    Returns:
        retList 支持度大于 minSupport 的集合
        supportData 候选项集支持度数据
    """

    # ssCnt 临时存放选数据集 Ck 的频率. 例如: a->10, b->5, c->8
    ssCnt = {}
    for tid in D:
        for can in Ck:
            # s.issubset(t)  测试是否 s 中的每一个元素都在 t 中
            if can.issubset(tid):
                if not ssCnt.has_key(can):
                    ssCnt[can] = 1
                else:
                    ssCnt[can] += 1
    numItems = float(len(D)) # 数据集 D 的数量
    retList = []
    supportData = {}
    for key in ssCnt:
        # 支持度 = 候选项（key）出现的次数 / 所有数据集的数量
        support = ssCnt[key]/numItems
        if support >= minSupport:
            # 在 retList 的首位插入元素，只存储支持度满足频繁项集的值
            retList.insert(0, key)
        # 存储所有的候选项（key）和对应的支持度（support）
        supportData[key] = support
    return retList, supportData
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/11.Apriori/apriori.py

组织完整的 Apriori 算法

输入频繁项集列表 Lk 与返回的元素个数 k，然后输出所有可能的候选项集 Ck

```
# 输入频繁项集列表 Lk 与返回的元素个数 k，然后输出所有可能的候选项集 Ck
def aprioriGen(Lk, k):
    """aprioriGen（输入频繁项集列表 Lk 与返回的元素个数 k，然后输出候选项集 Ck。
       例如: 以 {0},{1},{2} 为输入且 k = 2 则输出 {0,1}, {0,2}, {1,2}. 以 {0,1},{0,2},{1,2} 为输入且 k = 3 则输出 {0,1,2}
       仅需要计算一次，不需要将所有的结果计算出来，然后进行去重操作
       这是一个更高效的算法）

    Args:
        Lk 频繁项集列表
        k 返回的项集元素个数（若元素的前 k-2 相同，就进行合并）
    Returns:
        retList 元素两两合并的数据集
```

```
    """

    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            L1 = list(Lk[i])[: k-2]
            L2 = list(Lk[j])[: k-2]
            # print '-----i=', i, k-2, Lk, Lk[i], list(Lk[i])[: k-2]
            # print '-----j=', j, k-2, Lk, Lk[j], list(Lk[j])[: k-2]
            L1.sort()
            L2.sort()
            # 第一次 L1,L2 为空，元素直接进行合并，返回元素两两合并的数据集
            # if first k-2 elements are equal
            if L1 == L2:
                # set union
                # print 'union=', Lk[i] | Lk[j], Lk[i], Lk[j]
                retList.append(Lk[i] | Lk[j])
    return retList
```

找出数据集 dataSet 中支持度 >= 最小支持度的候选项集以及它们的支持度。即我们的频繁项集。

```
# 找出数据集 dataSet 中支持度 >= 最小支持度的候选项集以及它们的支持度。即我们的频繁项集。
def apriori(dataSet, minSupport=0.5):
    """apriori（首先构建集合 C1，然后扫描数据集来判断这些只有一个元素的项集是否满足最小支持度的要求。那么满足最小支持度要求的项集构成集合 L1。然

    Args:
        dataSet 原始数据集
        minSupport 支持度的阈值
    Returns:
        L 频繁项集的全集
        supportData 所有元素和支持度的全集
    """
    # C1 即对 dataSet 进行去重，排序，放入 list 中，然后转换所有的元素为 frozenset
    C1 = createC1(dataSet)
    # 对每一行进行 set 转换，然后存放到集合中
    D = map(set, dataSet)
    print 'D=', D
    # 计算候选数据集 C1 在数据集 D 中的支持度，并返回支持度大于 minSupport 的数据
    L1, supportData = scanD(D, C1, minSupport)
    # print "L1=", L1, "\n", "outcome: ", supportData

    # L 加了一层 list, L 一共 2 层 list
    L = [L1]
    k = 2
    # 判断 L 的第 k-2 项的数据长度是否 > 0。第一次执行时 L 为 [[frozenset([1]), frozenset([3]), frozenset([2]), frozenset([5])]]。L[k-2]=
    while (len(L[k-2]) > 0):
        print 'k=', k, L, L[k-2]
        Ck = aprioriGen(L[k-2], k) # 例如: 以 {0},{1},{2} 为输入且 k = 2 则输出 {0,1}, {0,2}, {1,2}. 以 {0,1},{0,2},{1,2} 为输入且 k = 3
        print 'Ck', Ck

        Lk, supK = scanD(D, Ck, minSupport) # 计算候选数据集 CK 在数据集 D 中的支持度，并返回支持度大于 minSupport 的数据
        # 保存所有候选项集的支持度，如果字典没有，就追加元素，如果有，就更新元素
        supportData.update(supK)
        if len(Lk) == 0:
            break
        # Lk 表示满足频繁子项的集合，L 元素在增加，例如:
        # l=[[set(1), set(2), set(3)]]
        # l=[[set(1), set(2), set(3)], [set(1, 2), set(2, 3)]]
        L.append(Lk)
        k += 1
        # print 'k=', k, len(L[k-2])
    return L, supportData
```

到这一步，我们就找出我们所需要的 频繁项集 和他们的 支持度 了，接下来再找出关联规则即可！

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/11.Apriori/apriori.py

从频繁项集中挖掘关联规则

前面我们介绍了用于发现 频繁项集 的 Apriori 算法，现在要解决的问题是如何找出 关联规则。

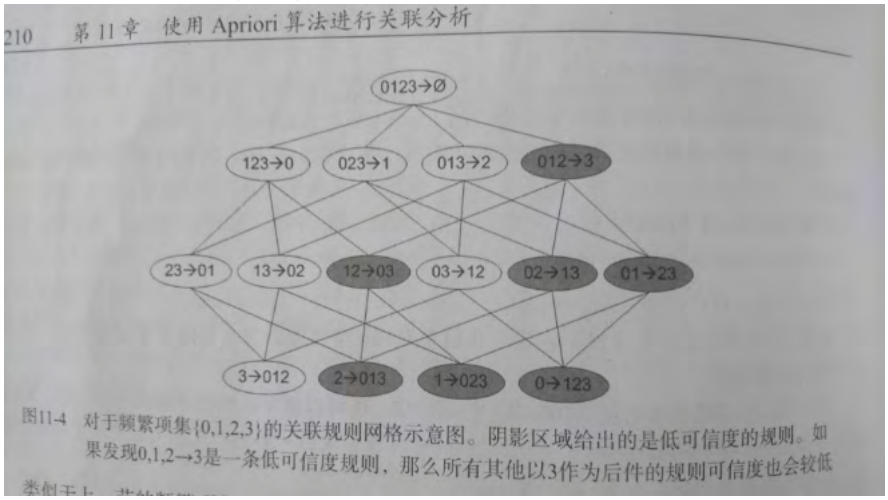
要找到 关联规则，我们首先从一个 频繁项集 开始。 我们知道集合中的元素是不重复的，但我们想知道基于这些元素能否获得其它内容。 某个元素或某个元素集合可能会推导出另一个元素。 从先前 杂货店 的例子可以得到，如果有一个频繁项集 {豆奶,莴苣}，那么就可能有一条关联规则 “豆奶 -> 莴苣”。 这意味着如果有人买了豆奶，那么在统计上他会购买莴苣的概率比较大。 但是，这一条件反过来并不总是成立。 也就是说 “豆奶 -> 莴苣” 统计上显著，那么 “莴苣 -> 豆奶” 也不一定成立。

前面我们给出了 频繁项集 的量化定义，即它满足最小支持度要求。
对于 关联规则，我们也有类似的量化方法，这种量化指标称之为 可信度。

一条规则 A -> B 的可信度定义为 support(A | B) / support(A)。（注意: 在 python 中 | 表示集合的并操作，而数学书集合并的符号是 U）。
`A | B` 是指所有出现在集合 A 或者集合 B 中的元素。
由于我们先前已经计算出所有 频繁项集 的支持度了，现在我们要做的只不过是提取这些数据做一次除法运算即可。

一个频繁项集可以产生多少条关联规则呢？

如下图所示，给出的是项集 {0,1,2,3} 产生的所有关联规则:



与我们前面的 频繁项集 生成一样，我们可以为每个频繁项集产生许多关联规则。
如果能减少规则的数目来确保问题的可解析，那么计算起来就会好很多。
通过观察，我们可以知道，如果某条规则并不满足 最小可信度 要求，那么该规则的所有子集也不会满足 最小可信度 的要求。
如上图所示，假设 `123 -> 3` 并不满足最小可信度要求，那么就知道任何左部为 {0,1,2} 子集的规则也不会满足 最小可信度 的要求。 即 `12 -> 03`, `02 -> 13`, `01 -> 23`, `2 -> 013`, `1 -> 023`, `0 -> 123` 都不满足 最小可信度 要求。

可以利用关联规则的上述性质属性来减少需要测试的规则数目，跟先前 Apriori 算法的套路一样。
以下是一些辅助函数:

计算可信度

```
# 计算可信度（confidence）
def calcConf(freqSet, H
, supportData, brl, minConf=0.7):
    """calcConf（对两个元素的频繁项，计算可信度，例如： {1,2}/{1} 或者 {1,2}/{2} 看是否满足条件）

    Args:
        freqSet 频繁项集中的元素，例如: frozenset([1, 3])
        H 频繁项集中的元素的集合，例如: [frozenset([1]), frozenset([3])]
        supportData 所有元素的支持度的字典
        brl 关联规则列表的空数组
        minConf 最小可信度
    Returns:
        prunedH 记录 可信度大于阈值的集合
    """
    # 记录可信度大于最小可信度（minConf）的集合
    prunedH = []
    for conseq in H: # 假设 freqSet = frozenset([1, 3]), H = [frozenset([1]), frozenset([3])]，那么现在需要求出 frozenset([1]) -> froze

        # print 'confData=', freqSet, H, conseq, freqSet-conseq
        conf = supportData[freqSet]/supportData[freqSet-conseq] # 支持度定义: a -> b = support(a | b) / support(a). 假设  freqSet = fr
        if conf >= minConf:
            # 只要买了 freqSet-conseq 集合，一定会买 conseq 集合（freqSet-conseq 集合和 conseq 集合是全集）
            print freqSet-conseq, '-->', conseq, 'conf:', conf
            brl.append((freqSet-conseq, conseq, conf))
            prunedH.append(conseq)
    return prunedH
```

递归计算频繁项集的规则

```
# 递归计算频繁项集的规则
def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    """rulesFromConseq

    Args:
        freqSet 频繁项集中的元素，例如: frozenset([2, 3, 5])
        H 频繁项集中的元素的集合，例如: [frozenset([2]), frozenset([3]), frozenset([5])]
        supportData 所有元素的支持度的字典
        brl 关联规则列表的数组
        minConf 最小可信度
    """
    # H[0] 是 freqSet 的元素组合的第一个元素，并且 H 中所有元素的长度都一样，长度由 aprioriGen(H, m+1) 这里的 m + 1 来控制
```

```
    # 该函数递归时，H[0] 的长度从 1 开始增长 1 2 3 ...
    # 假设 freqSet = frozenset([2, 3, 5]), H = [frozenset([2]), frozenset([3]), frozenset([5])]
    # 那么 m = len(H[0]) 的递归的值依次为 1 2
    # 在 m = 2 时，跳出该递归。假设再递归一次，那么 H[0] = frozenset([2, 3, 5])，freqSet = frozenset([2, 3, 5]) ，没必要再计算 freqSet 与
    m = len(H[0])
    if (len(freqSet) > (m + 1)):
        print 'freqSet******************', len(freqSet), m + 1, freqSet, H, H[0]
        # 生成 m+1 个长度的所有可能的 H 中的组合，假设 H = [frozenset([2]), frozenset([3]), frozenset([5])]
        # 第一次递归调用时生成 [frozenset([2, 3]), frozenset([2, 5]), frozenset([3, 5])]
        # 第二次 。。。没有第二次，递归条件判断时已经退出了
        Hmp1 = aprioriGen(H, m+1)
        # 返回可信度大于最小可信度的集合
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        print 'Hmp1=', Hmp1
        print 'len(Hmp1)=', len(Hmp1), 'len(freqSet)=', len(freqSet)
        # 计算可信度后，还有数据大于最小可信度的话，那么继续递归调用，否则跳出递归
        if (len(Hmp1) > 1):
            print '----------------------', Hmp1
            # print len(freqSet),  len(Hmp1[0]) + 1
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)
```

生成关联规则

```
# 生成关联规则
def generateRules(L, supportData, minConf=0.7):
    """generateRules

    Args:
        L 频繁项集列表
        supportData 频繁项集支持度的字典
        minConf 最小置信度
    Returns:
        bigRuleList 可信度规则列表（关于 (A->B+置信度) 3个字段的组合）
    """
    bigRuleList = []
    # 假设 L = [[frozenset([1]), frozenset([3]), frozenset([2]), frozenset([5])], [frozenset([1, 3]), frozenset([2, 5]), frozenset([2
    for i in range(1, len(L)):
        # 获取频繁项集中每个组合的所有元素
        for freqSet in L[i]:
            # 假设：freqSet= frozenset([1, 3]), H1=[frozenset([1]), frozenset([3])]
            # 组合总的元素并遍历子元素，并转化为 frozenset 集合，再存放到 list 列表中
            H1 = [frozenset([item]) for item in freqSet]
            # 2 个的组合，走 else, 2 个以上的组合，走 if
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList
```

到这里为止，通过调用 generateRules 函数即可得出我们所需的 关联规则。

- 分级法： 频繁项集->关联规则
  - 1.首先从一个频繁项集开始，接着创建一个规则列表，其中规则右部分只包含一个元素，然后对这个规则进行测试。
  - 2.接下来合并所有剩余规则来创建一个新的规则列表，其中规则右部包含两个元素。
  - 如下图：

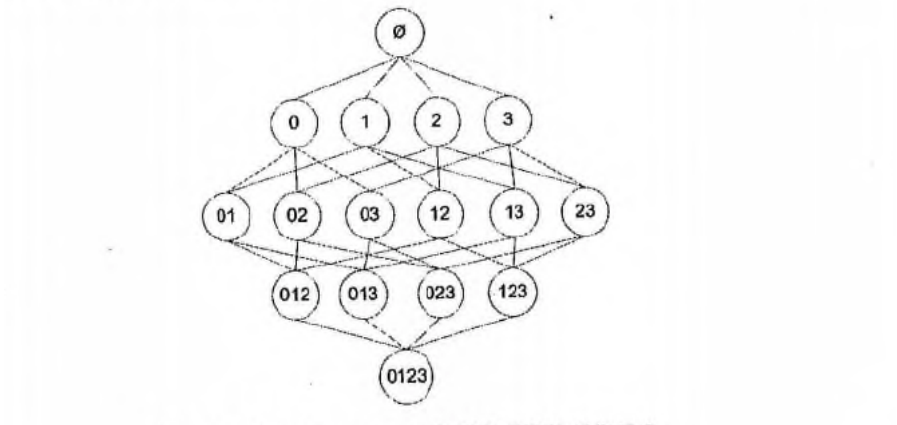


图11-2 集合{0，1，2，3}中所有可能的项集组合

- 最后： 每次增加频繁项集的大小，Apriori 算法都会重新扫描整个数据集，是否有优化空间呢？ 下一章：FP-growth算法等着你的到来

- 作者：片刻

- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第12章 使用FP-growth 算法来高效发现频繁项集

前言

在 第11章 时我们已经介绍了用 `Apriori` 算法发现 `频繁项集` 与 `关联规则`。
本章将继续关注发现 `频繁项集` 这一任务，并使用 `FP-growth` 算法更有效的挖掘 `频繁项集`。

FP-growth 算法简介

- 一种非常好的发现频繁项集算法。
- 基于Apriori算法构建,但是数据结构不同，使用叫做 `FP树` 的数据结构结构来存储集合。下面我们会介绍这种数据结构。

FP-growth 算法步骤

- 基于数据构建FP树
- 从FP树种挖掘频繁项集

FP树 介绍

- FP树的节点结构如下:

```
class treeNode:
    def __init__(self, nameValue, numOccur, parentNode):
        self.name = nameValue     # 节点名称
        self.count = numOccur     # 节点出现次数
        self.nodeLink = None      # 不同项集的相同项通过nodeLink连接在一起
        # needs to be updated
        self.parent = parentNode  # 指向父节点
        self.children = {}        # 存储叶子节点
```

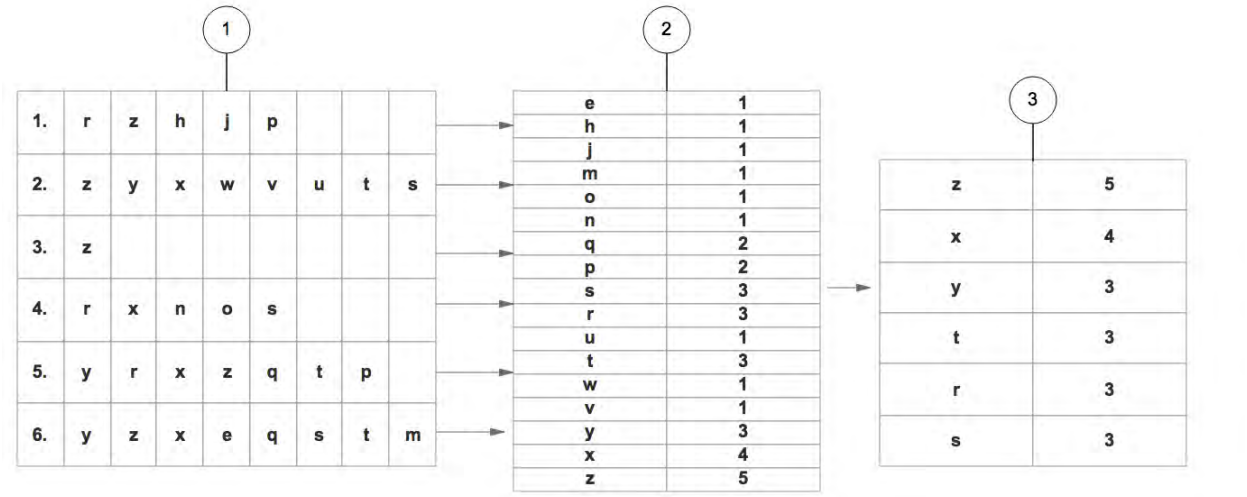
FP-growth 原理

基于数据构建FP树

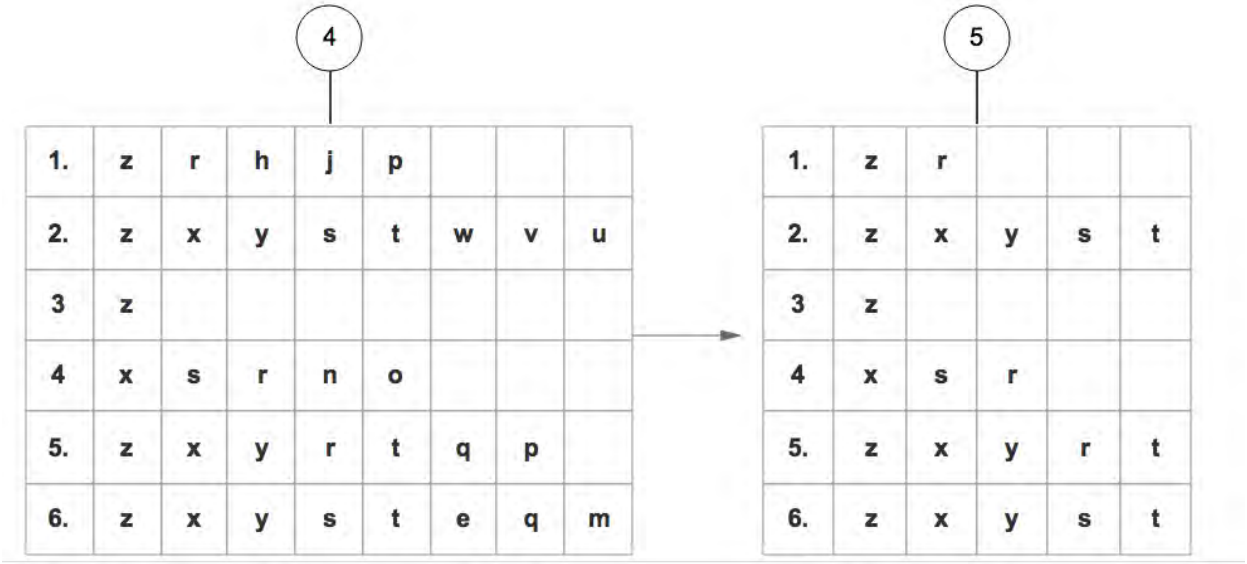
步骤1:

1. 遍历所有的数据集合，计算所有项的支持度。

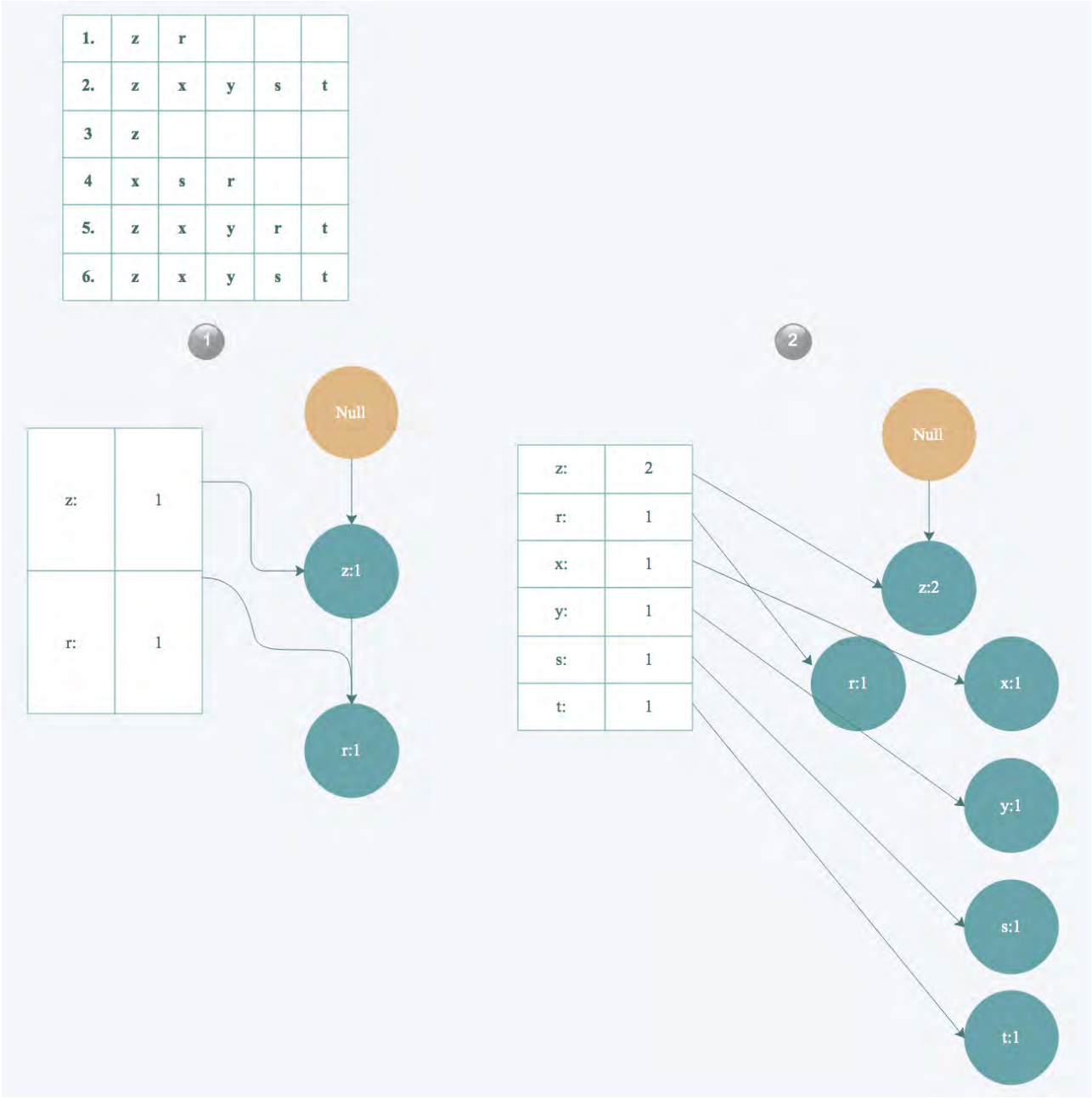
2. 丢弃非频繁的项。
3. 基于 支持度 降序排序所有的项。



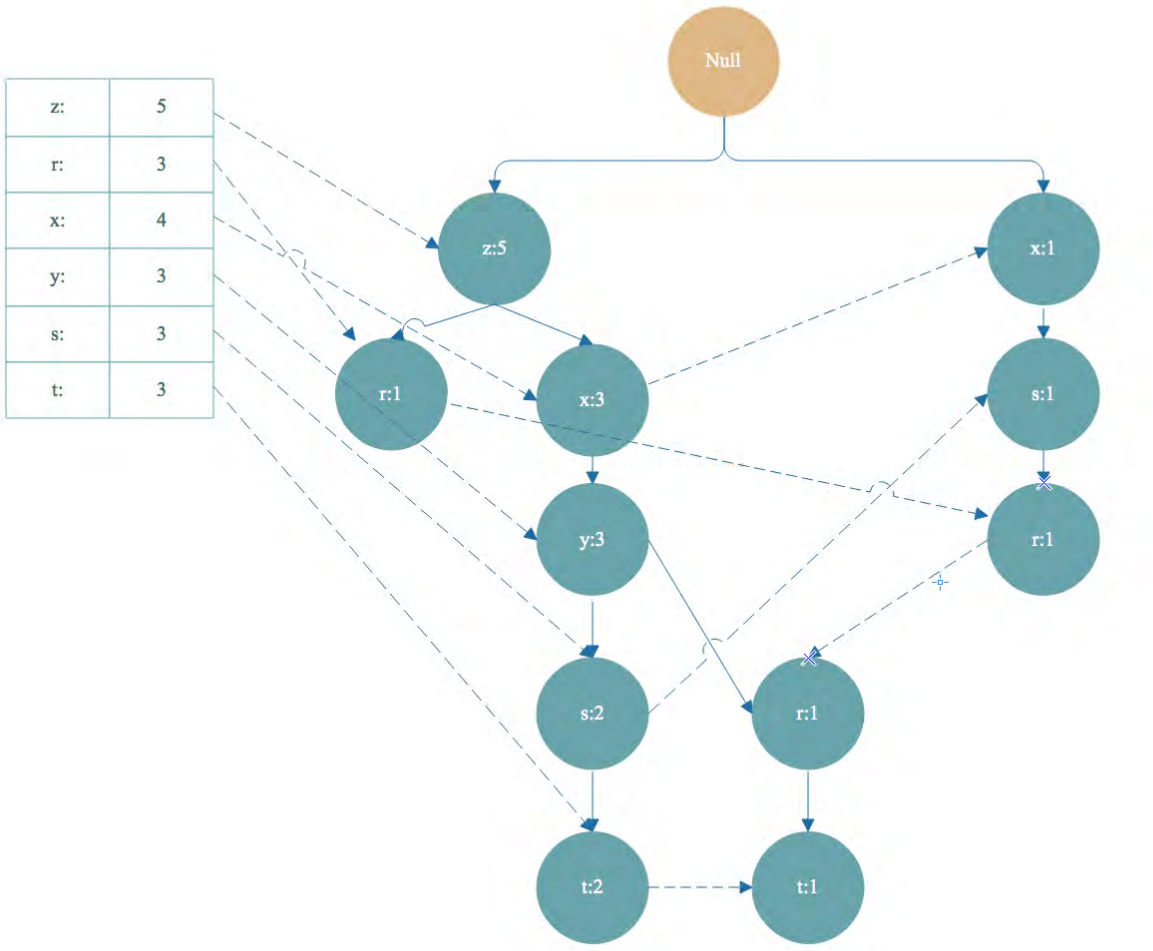
4. 所有数据集合按照得到的顺序重新整理。
5. 重新整理完成后，丢弃每个集合末尾非频繁的项。



步骤2: 6. 读取每个集合插入FP树中，同时用一个头部链表数据结构维护不同集合的相同项。



最终得到下面这样一棵FP树

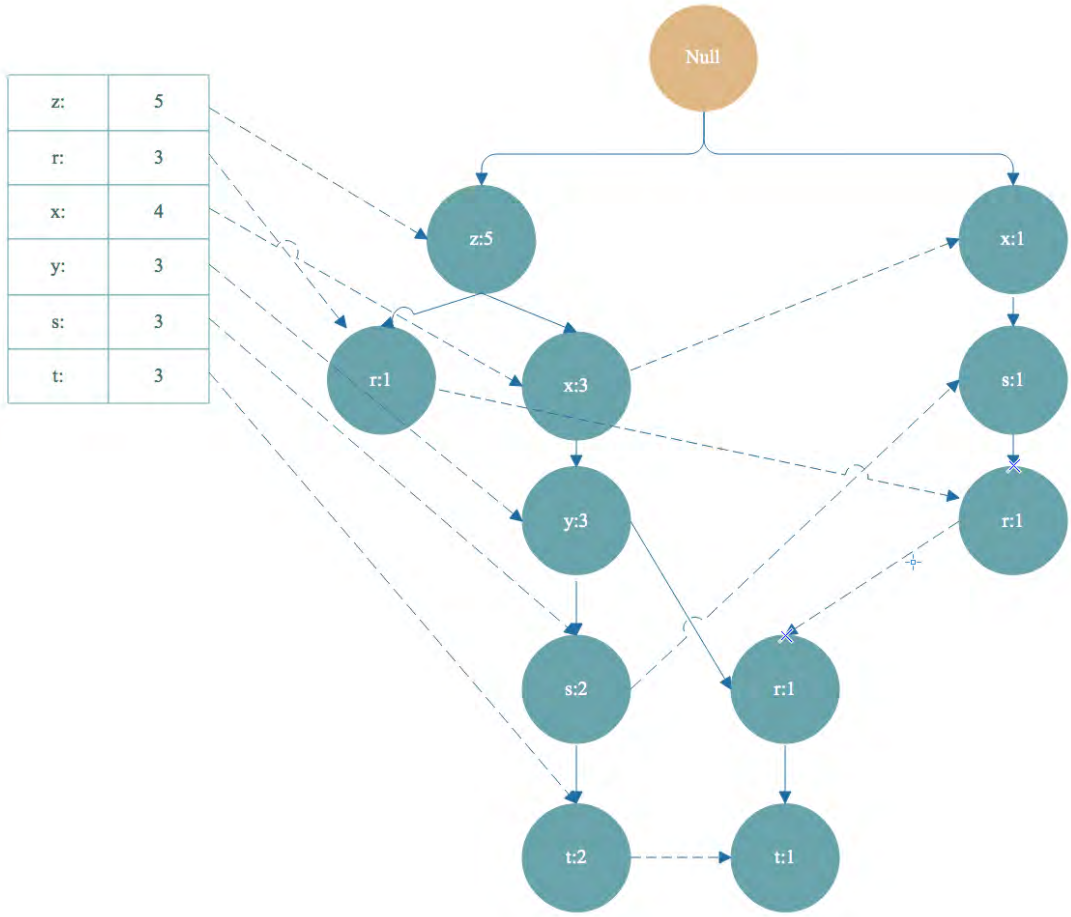


从FP树中挖掘出频繁项集

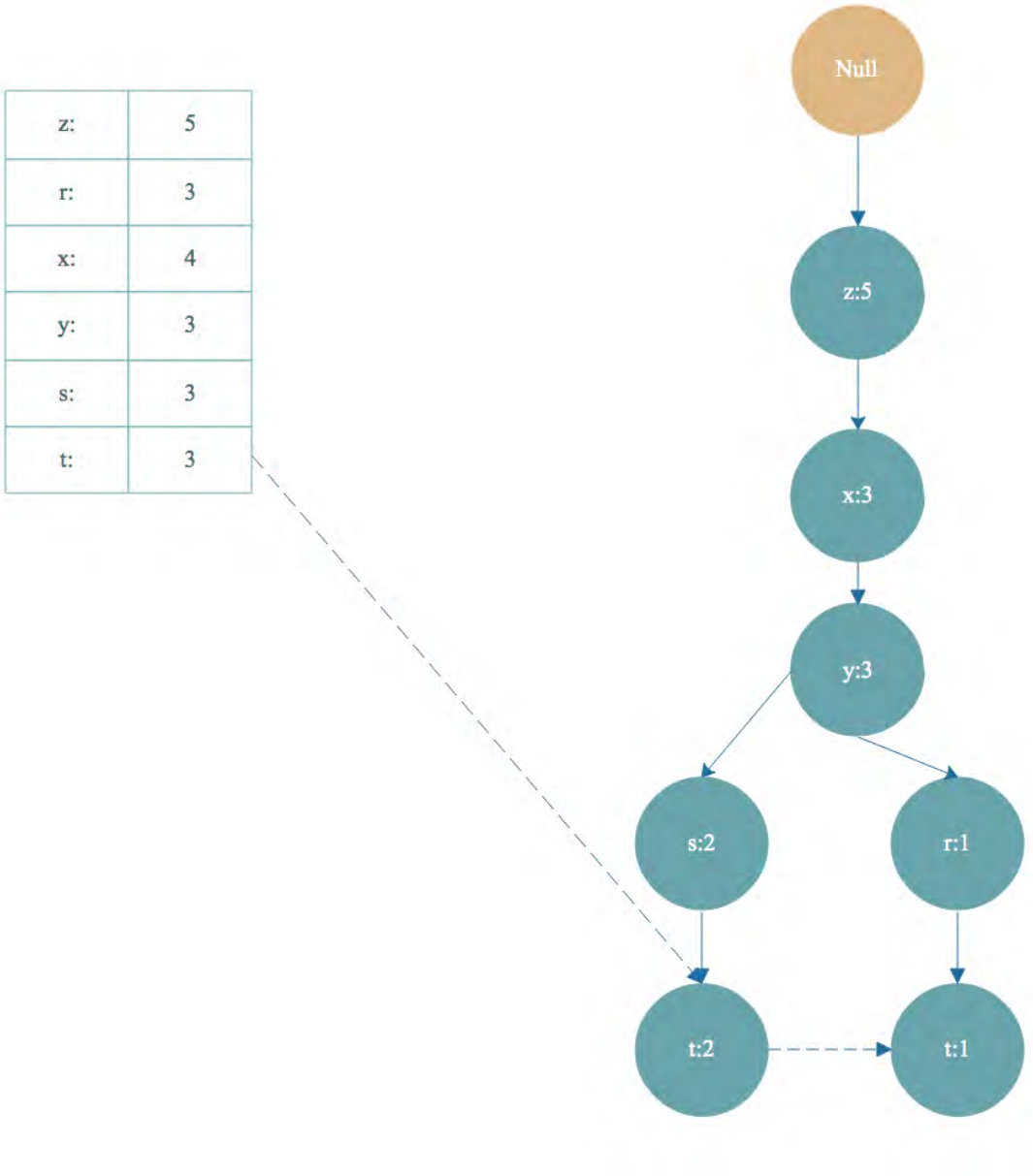
步骤3:

1. 对头部链表进行降序排序

2. 对头部链表节点从小到大遍历，得到条件模式基，同时获得一个频繁项集。

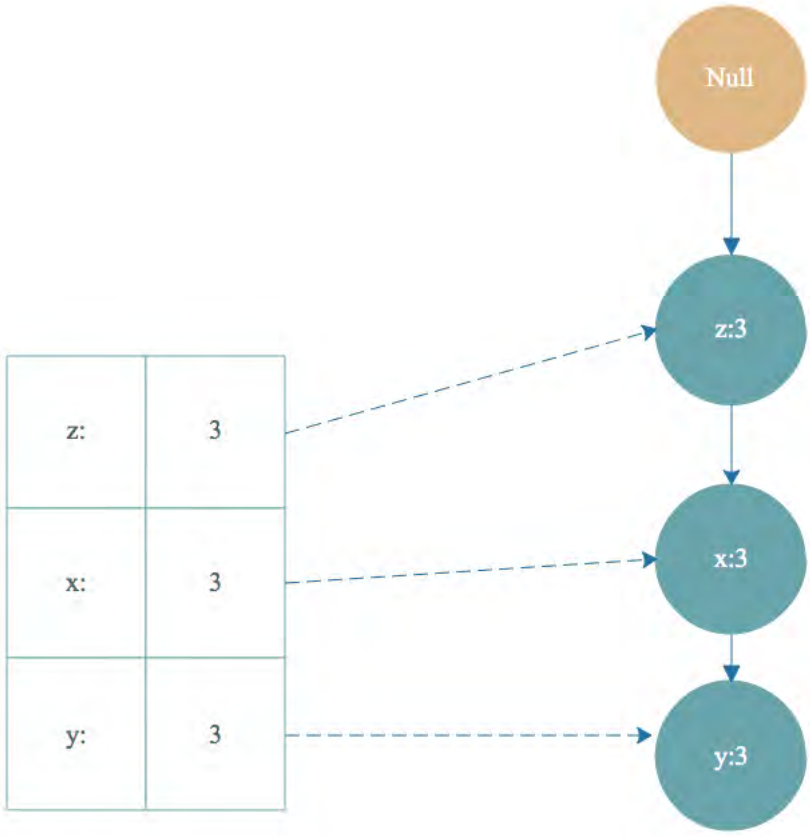


如上图，从头部链表 t 节点开始遍历，t 节点加入到频繁项集。找到以 t 节点为结尾的路径如下:



去掉FP树中的t节点，得到条件模式基<左边路径,左边是值>[z,x,y,s,t]:2，[z,x,y,r,t]:1 。条件模式基的值取决于末尾节点 t ，因为 t 的出现次数最小，一个频繁项集的支持度由支持度最小的项决定。所以 t 节点的条件模式基的值可以理解为对于以 t 节点为末尾的前缀路径出现次数。

3. 条件模式基继续构造条件 FP树， 得到频繁项集，和之前的频繁项组合起来，这是一个递归遍历头部链表生成FP树的过程，递归截止条件是生成的FP树的头部链表为空。 根据步骤 2 得到的条件模式基 [z,x,y,s,t]:2，[z,x,y,r,t]:1 作为数据集继续构造出一棵FP树，计算支持度，去除非频繁项，集合按照支持度降序排序，重复上面构造FP树的步骤。最后得到下面 t-条件FP树 :



然后根据 t-条件FP树 的头部链表进行遍历，从 y 开始。得到频繁项集 ty 。然后又得到 y 的条件模式基，构造出 ty的条件FP树，即 ty-条件FP树。继续遍历ty-条件FP树的头部链表，得到频繁项集 tyx，然后又得到频繁项集 tyxz. 然后得到构造tyxz-条件FP树的头部链表是空的，终止遍历。我们得到的频繁项集有 t->ty->tyz->tyzx，这只是一小部分。

- 条件模式基:头部链表中的某一点的前缀路径组合就是条件模式基，条件模式基的值取决于末尾节点的值。
- 条件FP树:以条件模式基为数据集构造的FP树叫做条件FP树。

FP-growth 算法优缺点:

```
* 优点： 1. 因为 FP-growth 算法只需要对数据集遍历两次，所以速度更快。
        2. FP树将集合按照支持度降序排序，不同路径如果有相同前缀路径共用存储空间，使得数据得到了压缩。
        3. 不需要生成候选集。
        4. 比Apriori更快。
* 缺点： 1. FP-Tree第二次遍历会存储很多中间过程的值，会占用很多内存。
        2. 构建FP-Tree是比较昂贵的。
* 适用数据类型：标称型数据(离散型数据)。
```

FP-growth 代码讲解

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/12.FrequentPattemTree/fpGrowth.py

main 方法大致步骤:

```
if __name__ == "__main__":
    simpDat = loadSimpDat()                         #加载数据集。
    initSet = createInitSet(simpDat)                #对数据集进行整理，相同集合进行合并。
    myFPtree, myHeaderTab = createTree(initSet, 3)#创建FP树。
    freqItemList = []
    mineTree(myFPtree, myHeaderTab, 3, set([]), freqItemList) #递归的从FP树中挖掘出频繁项集。
    print freqItemList
```

大家看懂原理，再仔细跟踪一下代码。基本就没有问题了。

- 作者：mikechengwei
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第13章 利用 PCA 来简化数据

降维技术

> 场景

- 我们正通过电视观看体育比赛，在电视的显示器上有一个球。
- 显示器大概包含了100万像素点，而球则可能是由较少的像素点组成，例如说一千个像素点。
- 人们实时的将显示器上的百万像素转换成为一个三维图像，该图像就给出运动场上球的位置。
- 在这个过程中，人们已经将百万像素点的数据，降至为三维。这个过程就称为`降维(dimensionality reduction)`

> 数据显示 并非大规模特征下的唯一难题，对数据进行简化还有如下一系列的原因：

- 1. 使得数据集更容易使用
- 2. 降低很多算法的计算开销
- 3. 去除噪音
- 4. 使得结果易懂

> 适用范围:

- 在已标注与未标注的数据上都有降维技术。
- 这里我们将主要关注未标注数据上的降维技术，将技术同样也可以应用于已标注的数据。

> 在以下3种降维技术中， PCA的应用目前最为广泛，因此本章主要关注PCA。

- 1. 主成分分析(Principal Component Analysis, PCA)
  - `通俗理解：就是找出一个最主要的特征，然后进行分析。`
  - `例如： 考察一个人的智力情况，就直接看数学成绩就行(存在：数学、语文、英语成绩)`
- 2. 因子分析(Factor Analysis)
  - `通俗理解：将多个实测变量转换为少数几个综合指标。它反映一种降维的思想，通过降维将相关性高的变量聚在一起,从而减少需要分析的变量的数量,而减少问题分析的复杂性`
  - `例如： 考察一个人的整体情况，就直接组合3样成绩(隐变量)，看平均成绩就行(存在：数学、语文、英语成绩)`
  - 应用的领域：社会科学、金融和其他领域
  - 在因子分析中，我们
    - 假设观察数据的成分中有一些观察不到的隐变量(latent variable)。
    - 假设观察数据是这些隐变量和某些噪音的线性组合。
    - 那么隐变量的数据可能比观察数据的数目少，也就说通过找到隐变量就可以实现数据的降维。

- 3. 独立成分分析(Independ Component Analysis, ICA)
    - 通俗理解：ICA 认为观测信号是若干个独立信号的线性组合，ICA 要做的是一个解混过程。
    - 例如：我们去ktv唱歌，想辨别唱的是什么歌曲？ICA 是观察发现是原唱唱的一首歌【2个独立的声音（原唱 / 主唱）】。
    - ICA 是假设数据是从 N 个数据源混合组成的，这一点和因子分析有些类似，这些数据源之间在统计上是相互独立的，而在 PCA 中只假设数据是不相关（线性关系）的。
    - 同因子分析一样，如果数据源的数目少于观察数据的数目，则可以实现降维过程。

PCA

PCA 概述

主成分分析(Principal Component Analysis, PCA)：通俗理解：就是找出一个最主要的特征，然后进行分析。

PCA 场景

例如： 考察一个人的智力情况，就直接看数学成绩就行(存在：数学、语文、英语成绩)

PCA 原理

> PCA 工作原理

1. 找出第一个主成分的方向，也就是数据 方差最大 的方向。
2. 找出第二个主成分的方向，也就是数据 方差次大 的方向，并且该方向与第一个主成分方向 正交(orthogonal 如果是二维空间就叫垂直)。
3. 通过这种方式计算出所有的主成分方向。
4. 通过数据集的协方差矩阵及其特征值分析，我们就可以得到这些主成分的值。
5. 一旦得到了协方差矩阵的特征值和特征向量，我们就可以保留最大的 N 个特征。这些特征向量也给出了 N 个最重要特征的真实结构，我们就可以通过将数据乘上这 N 个特征向量 从而将它转换到新的空间上。

例如下图：

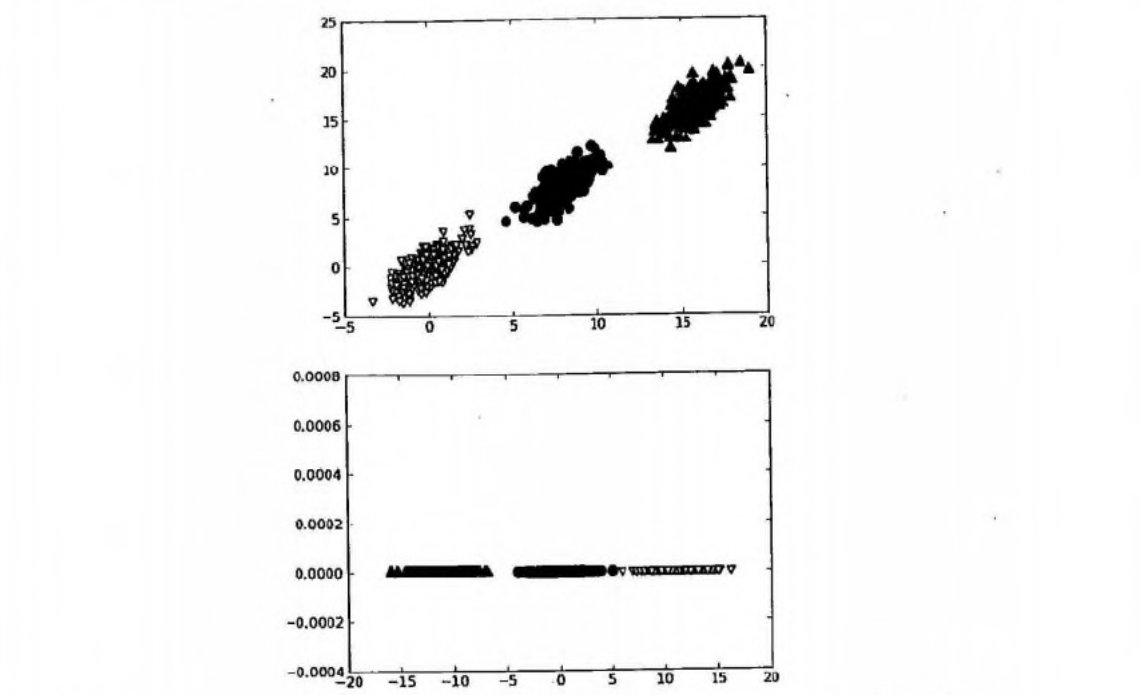


图13-2 二维空间的3个类别。当在该数据集上应用PCA时，就可以去掉一维，从而使得该分类问题变得更容易处理

> PCA 优缺点

```
通过 PCA 进行降维处理，我们就可以同时获得 SVM 和决策树的优点：(得到了和决策树一样简单的分类器，同时分类间隔和SVM一样好)
优点：降低数据的复杂性，识别最重要的多个特征。
缺点：不一定需要，且可能损失有用信息。
适用数据类型：数值型数据。
```

项目案例: 对半导体数据进行降维处理

项目概述

```
半导体是在一些极为先进的工厂中制造出来的。设备的生命早期有限，并且花费极其巨大。
虽然通过早期测试和频繁测试来发现有瑕疵的产品，但仍有一些存在瑕疵的产品通过测试。
如果我们通过机器学习技术用于发现瑕疵产品，那么它就会为制造商节省大量的资金。

具体来讲，它拥有590个特征。我们看看能否对这些特征进行降维处理。
```

对于数据的缺失值的问题，我们有一些处理方法(参考第5章)
目前该章节处理的方案是：将缺失值NaN(Not a Number缩写)，全部用平均值来替代(如果用0来处理的策略就太差劲了)。

开发流程

> 收集数据：提供文本文件

文件名：secom.data

文本文件数据格式如下：

```
3030.93 2564 2187.7333 1411.1265 1.3602 100 97.6133 0.1242 1.5005 0.0162 -0.0034 0.9455 202.4396 0 7.9558 414.871 10.0433 0.968 192.3963 12.519
3095.78 2465.14 2230.4222 1463.6606 0.8294 100 102.3433 0.1247 1.4966 -0.0005 -0.0148 0.9627 200.547 0 10.1548 414.7347 9.2599 0.9701 191.2872 1
2932.61 2559.94 2186.4111 1698.0172 1.5102 100 95.4878 0.1241 1.4436 0.0041 0.0013 0.9615 202.0179 0 9.5157 416.7075 9.3144 0.9674 192.7035 12.5
2988.72 2479.9 2199.0333 909.7926 1.3204 100 104.2367 0.1217 1.4882 -0.0124 -0.0033 0.9629 201.8482 0 9.6052 422.2894 9.6924 0.9687 192.1557 12.
3032.24 2502.87 2233.3667 1326.52 1.5334 100 100.3967 0.1235 1.5031 -0.0031 -0.0072 0.9569 201.9424 0 10.5661 420.5925 10.3387 0.9735 191.6037 1
```

> 准备数据：将value为NaN的求均值

```
def replaceNanWithMean():
    datMat = loadDataSet('input/13.PCA/secom.data', ' ')
    numFeat = shape(datMat)[1]
    for i in range(numFeat):
        # 对value不为NaN的求均值
        # .A 返回矩阵基于的数组
        meanVal = mean(datMat[nonzero(~isnan(datMat[:, i].A))[0], i])
        # 将value为NaN的值赋值为均值
        datMat[nonzero(isnan(datMat[:, i].A))[0],i] = meanVal
    return datMat
```

> 分析数据：统计分析 N 的阈值

表13-1给出了这些主成分所对应的方差百分比和累积方差百分比。浏览“累积方差百分比(%)”这一列就会注意到，前六个主成分就覆盖了数据96.8%的方差，而前20个主成分覆盖了99.3%的方差。这就表明了，如果保留前6个而去除后584个主成分，我们就可以实现大概100：1的压缩比。另外，由于舍弃了噪声的主成分，将后面的主成分去除便使得数据更加干净。

表13-1 半导体数据中前7个主成分所占的方差百分比

主 成 分	方差百分比（%）	累积方差百分比（%）
1	59.2	59.2
2	24.1	83.4
3	9.2	92.5
4	2.3	94.8
5	1.5	96.3
6	0.5	96.8
7	0.3	97.1
20	0.08	99.3

> PCA 数据降维

在等式 Av=入v 中，v 是特征向量， 入是特征值。
表示 如果特征向量 v 被某个矩阵 A 左乘，那么它就等于某个标量 入 乘以 v.
幸运的是： Numpy 中有寻找特征向量和特征值的模块 linalg，它有 eig() 方法，该方法用于求解特征向量和特征值。

```
def pca(dataMat, topNfeat=9999999):
    """pca

    Args:
        dataMat   原数据集矩阵
        topNfeat  应用的N个特征
    Returns:
        lowDDataMat  降维后数据集
        reconMat     新的数据集空间
    """

    # 计算每一列的均值
    meanVals = mean(dataMat, axis=0)
    # print 'meanVals', meanVals
```

```
    # 每个向量同时都减去 均值
    meanRemoved = dataMat - meanVals
    # print 'meanRemoved=', meanRemoved

    # cov协方差=[(x1-x均值)*(y1-y均值)+(x2-x均值)*(y2-y均值)+...+(xn-x均值)*(yn-y均值)+]/(n-1)
    '''
    方差：（一维）度量两个随机变量关系的统计量
    协方差： （二维）度量各个维度偏离其均值的程度
    协方差矩阵：（多维）度量各个维度偏离其均值的程度

    当 cov(X, Y)>0时，表明X与Y正相关；(X越大，Y也越大；X越小Y，也越小。这种情况，我们称为“正相关”。)
    当 cov(X, Y)<0时，表明X与Y负相关；
    当 cov(X, Y)=0时，表明X与Y不相关。
    '''
    covMat = cov(meanRemoved, rowvar=0)

    # eigVals为特征值， eigVects为特征向量
    eigVals, eigVects = linalg.eig(mat(covMat))
    # print 'eigVals=', eigVals
    # print 'eigVects=', eigVects
    # 对特征值，进行从小到大的排序，返回从小到大的index序号
    # 特征值的逆序就可以得到topNfeat个最大的特征向量
    '''
    >>> x = np.array([3, 1, 2])
    >>> np.argsort(x)
    array([1, 2, 0])  # index,1 = 1; index,2 = 2; index,0 = 3
    >>> y = np.argsort(x)
    >>> y[::-1]
    array([0, 2, 1])
    >>> y[:-3:-1]
    array([0, 2])  # 取出 -1, -2
    >>> y[:-6:-1]
    array([0, 2, 1])
    '''
    eigValInd = argsort(eigVals)
    # print 'eigValInd1=', eigValInd

    # -1表示倒序，返回topN的特征值[-1 到 -(topNfeat+1) 但是不包括-(topNfeat+1)本身的倒叙]
    eigValInd = eigValInd[:-(topNfeat+1):-1]
    # print 'eigValInd2=', eigValInd
    # 重组 eigVects 最大到最小
    redEigVects = eigVects[:, eigValInd]
    # print 'redEigVects=', redEigVects.T
    # 将数据转换到新空间
    # --- (1567, 590) (590, 20)
    # print "---", shape(meanRemoved), shape(redEigVects)
    lowDDataMat = meanRemoved * redEigVects
    reconMat = (lowDDataMat * redEigVects.T) + meanVals
    # print 'lowDDataMat=', lowDDataMat
    # print 'reconMat=', reconMat
    return lowDDataMat, reconMat
```

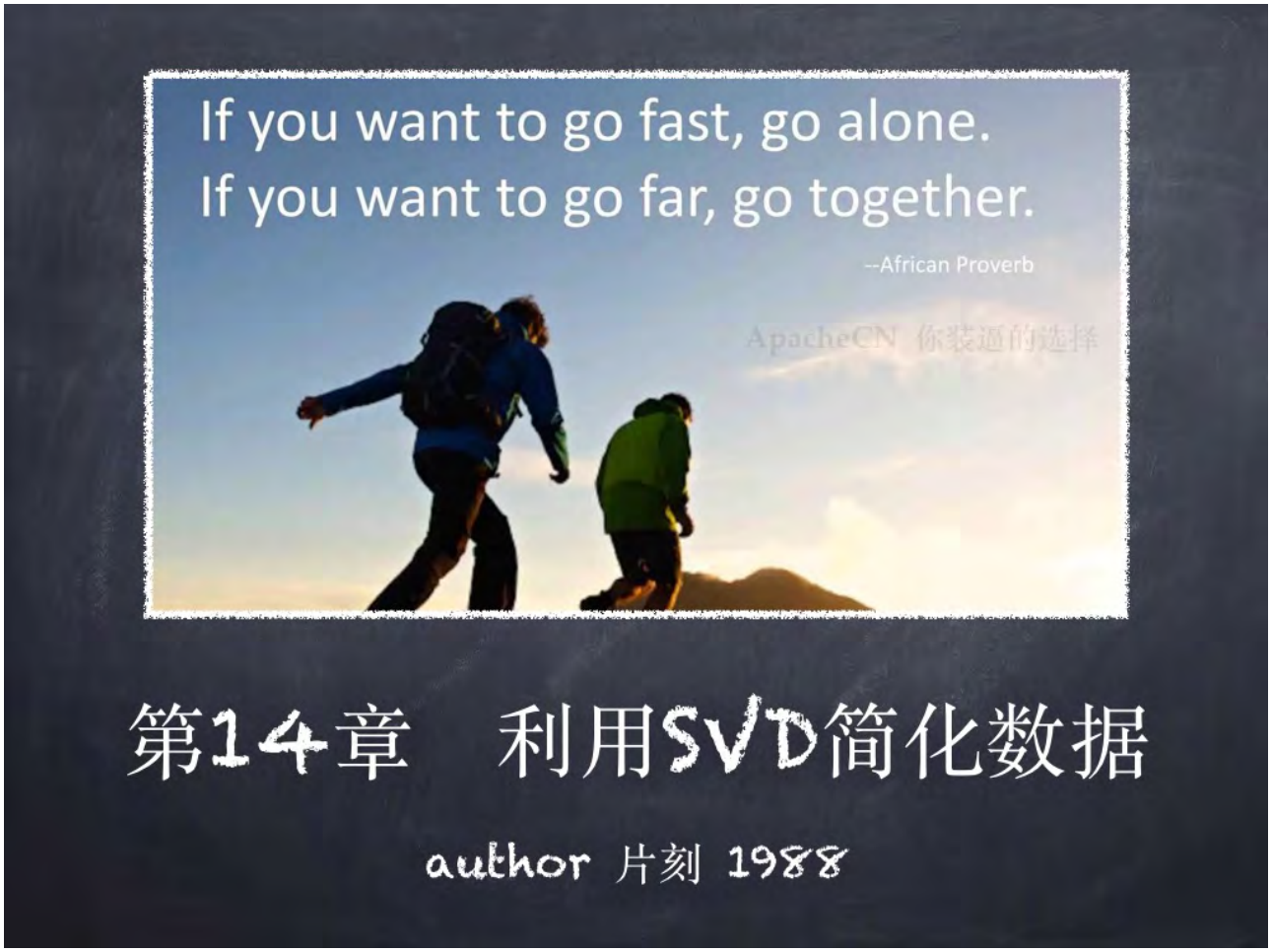
完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/13.PCA/pca.py

要点补充

```
降维技术使得数据变的更易使用，并且它们往往能够去除数据中的噪音，使得其他机器学习任务更加精确。
降维往往作为预处理步骤，在数据应用到其他算法之前清洗数据。
比较流行的降维技术： 独立成分分析、因子分析 和 主成分分析， 其中又以主成分分析应用最广泛。

本章中的PCA将所有的数据集都调入了内存，如果无法做到，就需要其他的方法来寻找其特征值。
如果使用在线PCA分析的方法，你可以参考一篇优秀的论文 "Incremental Eigenanalysis for Classification"。
下一章要讨论的奇异值分解方法也可以用于特征值分析。
```

- 作者：片刻 1988
- GitHub地址: https://github.com/apachecn/MachineLearning

第14章 利用SVD简化数据



SVD 概述

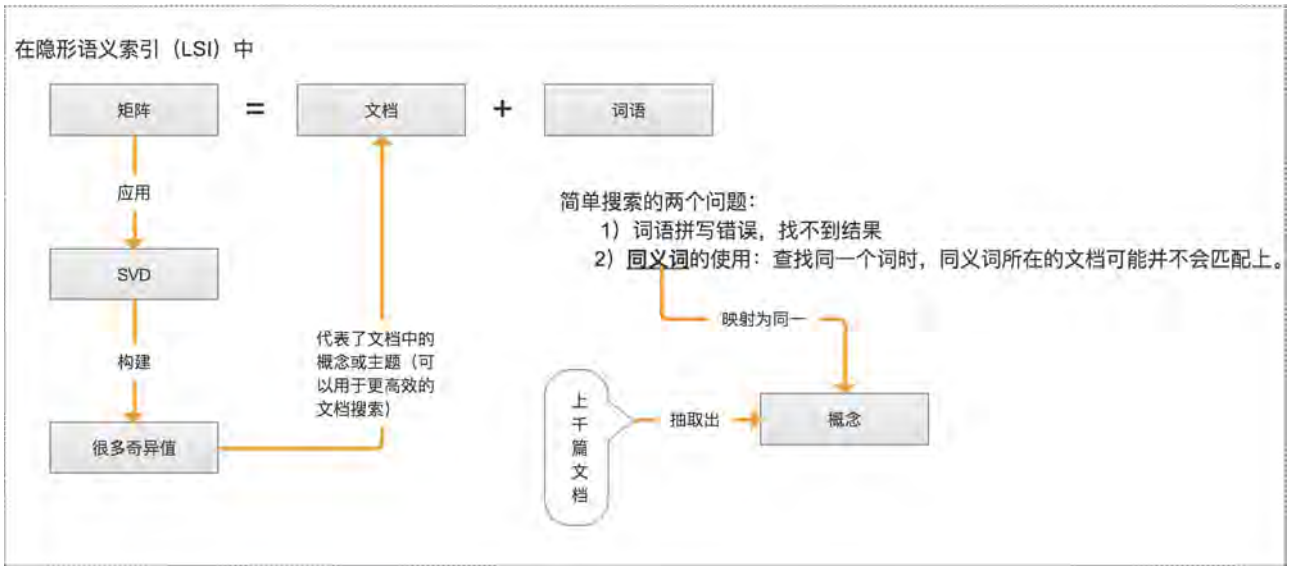
```
奇异值分解（SVD, Singular Value Decomposition）：
    提取信息的一种方法，可以把 SVD 看成是从噪声数据中抽取相关特征。从生物信息学到金融学，SVD 是提取信息的强大工具。
```

SVD 场景

> 信息检索-隐形语义检索（Lstent Semantic Indexing, LSI）或 隐形语义分析（Latent Semantic Analysis, LSA）

隐性语义索引：矩阵 = 文档 + 词语

- 是最早的 SVD 应用之一，我们称利用 SVD 的方法为隐性语义索引（LSI）或隐性语义分析（LSA）。



> 推荐系统

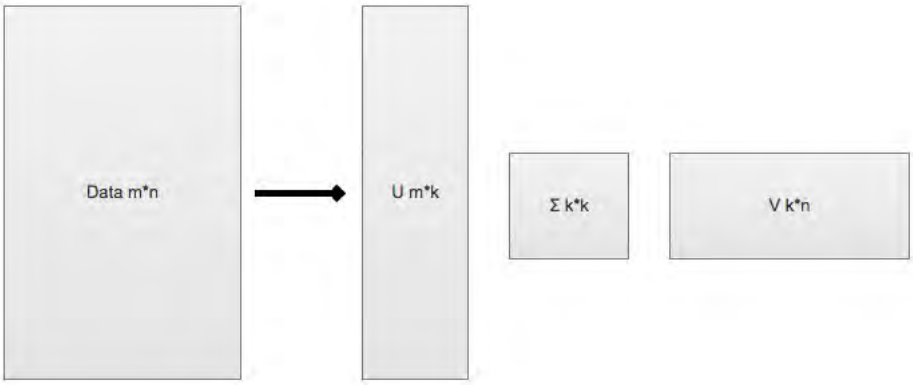
1. 利用 SVD 从数据中构建一个主题空间。
2. 再在该空间下计算其相似度。(从高维-低维空间的转化，在低维空间来计算相似度，SVD 提升了推荐系统的效率。)

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Ed	0	0	0	2	2
Peter	0	0	0	3	3
Tracy	0	0	0	1	1
Fan	1	1	1	0	0
Ming	2	2	2	0	0
Pachi	5	5	5	0	0
Jocelyn	1	1	1	0	0

- 上图右边标注的为一组共同特征，表示美式 BBQ 空间；另一组在上图右边未标注的为日式食品 空间。

> 图像压缩

例如：`32*32=1024 => 32*2+2*1+32*2=130`(2*1表示去掉了除对角线的0), 几乎获得了10倍的压缩比。



SVD 原理

SVD 工作原理

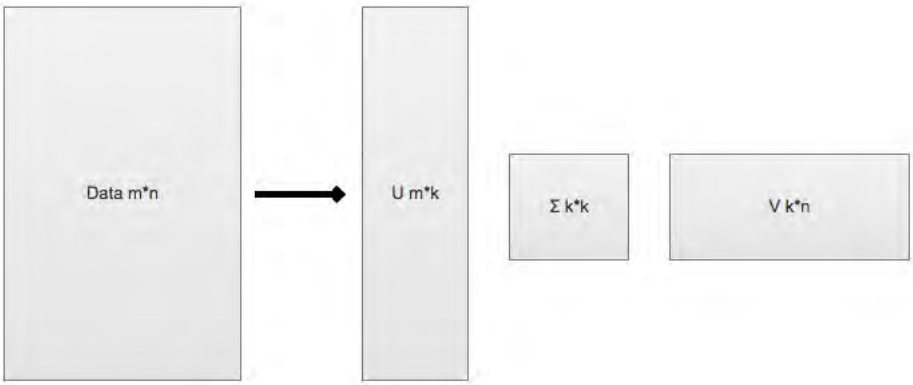
> 矩阵分解

- 矩阵分解是将数据矩阵分解为多个独立部分的过程。
- 矩阵分解可以将原始矩阵表示成新的易于处理的形式，这种新形式是两个或多个矩阵的乘积。（类似代数中的因数分解）
- 举例：如何将12分解成两个数的乘积？（1，12）、（2，6）、（3，4）都是合理的答案。

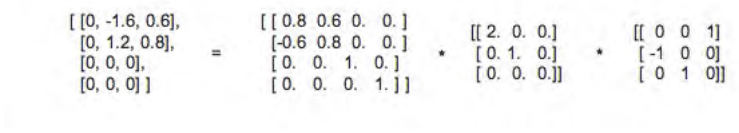
> SVD 是矩阵分解的一种类型，也是矩阵分解最常见的技术

- SVD 将原始的数据集矩阵 Data 分解成三个矩阵 U、∑、V
- 举例：如果原始矩阵 $Data_{m*n}$ 是m行n列，
  - U_{m*n} 表示m行n列
  - $\sum_{m*k}$ 表示m行k列
  - V_{k*n} 表示k行n列。

$$Data_{m*n} = U_{m*k} * \sum_{k*k} * V_{k*n}$$



具体的案例：（大家可以试着推导一下：https://wenku.baidu.com/view/b7641217866fb84ae45c8d17.html ）



- 上述分解中会构建出一个矩阵∑，该矩阵只有对角元素，其他元素均为0(近似于0)。另一个惯例就是，∑的对角元素是从大到小排列的。这些对角元素称为奇异值。
- 奇异值与特征值(PCA 数据中重要特征)是有关系的。这里的奇异值就是矩阵 $Data * Data^T$ 特征值的平方根。
- 普遍的事实：在某个奇异值的数目(r 个=>奇异值的平方和累加到总值的90%以上)之后，其他的奇异值都置为0(近似于0)。这意味着数据集中仅有 r 个重要特征，而其余特征则都是噪声或冗余特征。

SVD 算法特点

```
优点：简化数据，去除噪声，优化算法的结果
缺点：数据的转换可能难以理解
使用的数据类型：数值型数据
```

推荐系统

推荐系统 概述

推荐系统是利用电子商务网站向客户提供商品信息和建议，帮助用户决定应该购买什么产品，模拟销售人员帮助客户完成购买过程。

推荐系统 场景

1. Amazon 会根据顾客的购买历史向他们推荐物品
2. Netflix 会向其用户推荐电影
3. 新闻网站会对用户推荐新闻频道

推荐系统 要点

> 基于协同过滤(collaborative filtering) 的推荐引擎

- 利用Python 实现 SVD(Numpy 有一个称为 linalg 的线性代数工具箱)
- 协同过滤：是通过将用户和其他用户的数据进行对比来实现推荐的。
- 当知道了两个用户或两个物品之间的相似度，我们就可以利用已有的数据来预测未知用户的喜好。

> 基于物品的相似度和基于用户的相似度：物品比较少则选择物品相似度，用户比较少则选择用户相似度。【矩阵还是小一点好计算】

- 基于物品的相似度：计算物品之间的距离。【耗时会随物品数量的增加而增加】
- 由于物品A和物品C 相似度(相关度)很高，所以给买A的人推荐C。

用户/物品	物品A	物品B	物品C
用户A	√		√
用户B	√	√	√
用户C	√		推荐

- 基于用户的相似度：计算用户之间的距离。【耗时会随用户数量的增加而增加】
- 由于用户A和用户C 相似度(相关度)很高，所以A和C是兴趣相投的人，对于C买的物品就会推荐给A。

用户/物品	物品A	物品B	物品C	物品D
用户A	√		√	推荐
用户B		√		
用户C	√		√	√

> 相似度计算

- inA, inB 对应的是 列向量

1. 欧氏距离：指在m维空间中两个点之间的真实距离，或者向量的自然长度（即改点到原点的距离）。二维或三维中的欧氏距离就是两点之间的实际距离。
   - 相似度= 1/(1+欧式距离)
   - `相似度= 1.0/(1.0 + la.norm(inA - inB))`
   - 物品对越相似，它们的相似度值就越大。
2. 皮尔逊相关系数：度量的是两个向量之间的相似度。
   - 相似度= 0.5 + 0.5*corrcoef() 【皮尔逊相关系数的取值范围从 -1 到 +1，通过函数0.5 + 0.5*corrcoef()这个函数计算，把值归一化到0到1之间】
   - `相似度= 0.5 + 0.5 * corrcoef(inA, inB, rowvar = 0)[0][1]`
   - 相对欧氏距离的优势：它对用户评级的量级并不敏感。
3. 余弦相似度：计算的是两个向量夹角的余弦值。
   - 余弦值 = (A·B)/(||A||·||B||) 【余弦值的取值范围也在-1到+1之间】
   - 相似度= 0.5 + 0.5*余弦值
   - `相似度= 0.5 + 0.5*( float(inA.T*inB) / la.norm(inA)*la.norm(inB))`
   - 如果夹角为90度，则相似度为0；如果两个向量的方向相同，则相似度为1.0。

> 推荐系统的评价

- 采用交叉测试的方法。【拆分数据为训练集和测试集】
- 推荐引擎评价的指标： 最小均方根误差(Root mean squared error, RMSE)，也称标准误差(Standard error)，就是计算均方误差的平均值然后取其平方根。
  - 如果RMSE=1, 表示相差1个星级；如果RMSE=2.5, 表示相差2.5个星级。

推荐系统 原理

- 推荐系统的工作过程：给定一个用户，系统会为此用户返回N个最好的推荐菜。
- 实现流程大致如下：
  1. 寻找用户没有评级的菜肴，即在用户-物品矩阵中的0值。
  2. 在用户没有评级的所有物品中，对每个物品预计一个可能的评级分数。这就是说：我们认为用户可能会对物品的打分（这就是相似度计算的初衷）。
  3. 对这些物品的评分从高到低进行排序，返回前N个物品。

项目案例: 餐馆菜肴推荐系统

项目概述

`假如一个人在家决定外出吃饭，但是他并不知道该到哪儿去吃饭，该点什么菜。推荐系统可以帮他做到这两点。`

开发流程

> 收集 并 准备数据

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	三文鱼汉堡	鲁宾三明治	印度烤鸡	麻婆豆腐	宫保鸡丁	印度奶酪咖喱	俄式汉堡
Brett	2	0	0	4	4	0	0	0	0	0	0
Rob	0	0	0	0	0	0	0	0	0	0	5
Drew	0	0	0	0	0	0	0	1	0	4	0
Scott	3	3	4	0	3	0	0	2	2	0	0
Mary	5	5	5	0	0	0	0	0	0	0	0
Brent	0	0	0	0	0	0	5	0	0	5	0
Kyle	4	0	4	0	0	0	0	0	0	0	5
Sara	0	0	0	0	0	4	0	0	0	0	4
Shaney	0	0	0	0	0	0	5	0	0	5	0
Brendan	0	0	0	3	0	0	0	0	4	5	0
Leanna	1	1	2	1	1	2	1	0	4	5	0

图14-4 一个更大的用户－菜肴矩阵，其中有很多物品都没有评分，这比一个全填充的矩阵更接近真实情况

```
def loadExData3():
    # 利用SVD提高推荐效果，菜肴矩阵
    """
    行：代表人
    列：代表菜肴名词
    值：代表人对菜肴的评分，0表示未评分
    """
    return[[2, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5],
           [0, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0],
           [3, 3, 4, 0, 3, 0, 0, 2, 2, 0, 0],
```

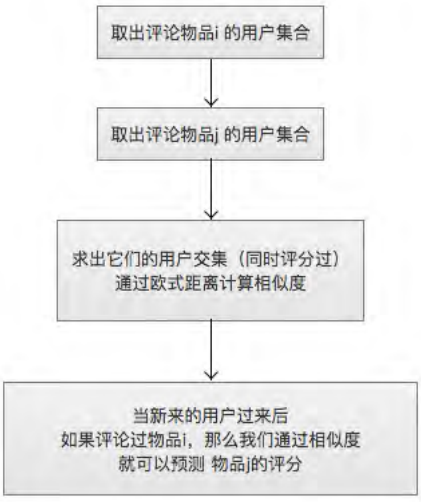
```
          [5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 5, 0, 0, 5, 0],
          [4, 0, 4, 0, 0, 0, 0, 0, 0, 0, 5],
          [0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 4],
          [0, 0, 0, 0, 0, 0, 5, 0, 0, 5, 0],
          [0, 0, 0, 3, 0, 0, 0, 0, 4, 5, 0],
          [1, 1, 2, 1, 1, 2, 1, 0, 4, 5, 0]]
```

> 分析数据: 这里不做过多的讨论(当然此处可以对比不同距离之间的差别)

> 训练算法: 通过调用 recommend() 函数进行推荐

recommend() 会调用 基于物品相似度 或者是 基于SVD，得到推荐的物品评分。

- 1.基于物品相似度



	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Jim	2	0	0	4	4
John	5	5	5	3	3
Sally	2	4	2	1	2

图14-3　用于展示相似度计算的简单矩阵

我们计算一下手撕猪肉和烤牛肉之间的相似度。一开始我们使用欧氏距离来计算。手撕猪肉和烤牛肉的欧氏距离为：

$$\sqrt{(4-4)^2+(3-3)^2+(2-1)^2}=1$$

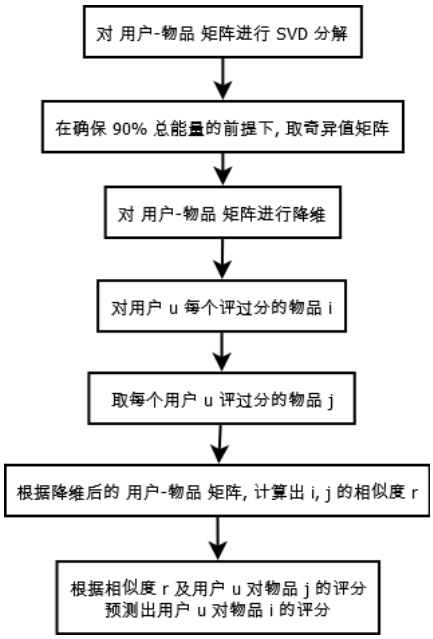
而手撕猪肉和鳗鱼饭的欧氏距离为：

$$\sqrt{(4-2)^2+(3-5)^2+(2-2)^2}=2.83$$

```
# 基于物品相似度的推荐引擎
def standEst(dataMat, user, simMeas, item):
    """standEst(计算某用户未评分物品中，以对该物品和其他物品评分的用户的物品相似度，然后进行综合评分)

    Args:
        dataMat         训练数据集
        user            用户编号
        simMeas         相似度计算方法
        item            未评分的物品编号
    Returns:
        ratSimTotal/simTotal     评分（0～5之间的值）
    """
    # 得到数据集中的物品数目
    n = shape(dataMat)[1]
    # 初始化两个评分值
    simTotal = 0.0
```

```
    ratSimTotal = 0.0
    # 遍历行中的每个物品（对用户评过分的物品进行遍历，并将它与其他物品进行比较）
    for j in range(n):
        userRating = dataMat[user, j]
        # 如果某个物品的评分值为0，则跳过这个物品
        if userRating == 0:
            continue
        # 寻找两个用户都评级的物品
        # 变量 overLap 给出的是两个物品当中已经被评分的那个元素的索引ID
        # logical_and 计算x1和x2元素的真值。
        overLap = nonzero(logical_and(dataMat[:, item].A > 0, dataMat[:, j].A > 0))[0]
        # 如果相似度为0，则两着没有任何重合元素，终止本次循环
        if len(overLap) == 0:
            similarity = 0
        # 如果存在重合的物品，则基于这些重合物重新计算相似度。
        else:
            similarity = simMeas(dataMat[overLap, item], dataMat[overLap, j])
        # print 'the %d and %d similarity is : %f'(iten,j,similarity)
        # 相似度会不断累加，每次计算时还考虑相似度和当前用户评分的乘积
        # similarity  用户相似度,    userRating 用户评分
        simTotal += similarity
        ratSimTotal += similarity * userRating
    if simTotal == 0:
        return 0
    # 通过除以所有的评分总和，对上述相似度评分的乘积进行归一化，使得最后评分在0~5之间，这些评分用来对预测值进行排序
    else:
        return ratSimTotal/simTotal
```

- 2.基于SVD(参考地址：http://www.codeweblog.com/svd-%E7%AC%94%E8%AE%B0/)




```
# 基于SVD的评分估计
# 在recommend() 中，这个函数用于替换对standEst()的调用，该函数对给定用户给定物品构建了一个评分估计值
def svdEst(dataMat, user, simMeas, item):
    """svdEst(计算某用户未评分物品中，以对该物品和其他物品评分的用户的物品相似度，然后进行综合评分)

    Args:
        dataMat         训练数据集
        user            用户编号
        simMeas         相似度计算方法
        item            未评分的物品编号
    Returns:
        ratSimTotal/simTotal     评分（0～5之间的值）
    """
    # 物品数目
    n = shape(dataMat)[1]
    # 对数据集进行SVD分解
    simTotal = 0.0
    ratSimTotal = 0.0
    # 奇异值分解
    # 在SVD分解之后，我们只利用包含了90%能量值的奇异值，这些奇异值会以NumPy数组的形式得以保存
    U, Sigma, VT = la.svd(dataMat)

    # # 分析 Sigma 的长度取值
    # analyse_data(Sigma, 20)

    # 如果要进行矩阵运算，就必须要用这些奇异值构建出一个对角矩阵
```

```
    Sig4 = mat(eye(4) * Sigma[: 4])
    # 利用U矩阵将物品转换到低维空间中，构建转换后的物品(物品+4个主要的特征)
    xformedItems = dataMat.T * U[:, :4] * Sig4.I
    # 对于给定的用户，for循环在用户对应行的元素上进行遍历，
    # 这和standEst()函数中的for循环的目的一样，只不过这里的相似度计算时在低维空间下进行的。
    for j in range(n):
        userRating = dataMat[user, j]
        if userRating == 0 or j == item:
            continue
        # 相似度的计算方法也会作为一个参数传递给该函数
        similarity = simMeas(xformedItems[item, :].T, xformedItems[j, :].T)
        # for 循环中加入了一条print语句，以便了解相似度计算的进展情况。如果觉得累赘，可以去掉
        print 'the %d and %d similarity is: %f' % (item, j, similarity)
        # 对相似度不断累加求和
        simTotal += similarity
        # 对相似度及对应评分值的乘积求和
        ratSimTotal += similarity * userRating
    if simTotal == 0:
        return 0
    else:
        # 计算估计评分
        return ratSimTotal/simTotal
```

排序获取最后的推荐结果

```
# recommend()函数，就是推荐引擎，它默认调用standEst()函数，产生了最高的N个推荐结果。
# 如果不指定N的大小，则默认值为3。该函数另外的参数还包括相似度计算方法和估计方法
def recommend(dataMat, user, N=3, simMeas=cosSim, estMethod=standEst):
    # 寻找未评级的物品
    # 对给定的用户建立一个未评分的物品列表
    unratedItems = nonzero(dataMat[user, :].A == 0)[1]
    # 如果不存在未评分物品，那么就退出函数
    if len(unratedItems) == 0:
        return 'you rated everything'
    # 物品的编号和评分值
    itemScores = []
    # 在未评分物品上进行循环
    for item in unratedItems:
        estimatedScore = estMethod(dataMat, user, simMeas, item)
        # 寻找前N个未评级物品，调用standEst()来产生该物品的预测得分，该物品的编号和估计值会放在一个元素列表itemScores中
        itemScores.append((item, estimatedScore))
        # 按照估计得分，对该列表进行排序并返回。列表逆排序，第一个值就是最大值
    return sorted(itemScores, key=lambda jj: jj[1], reverse=True)[: N]
```

> 测试 和 项目调用，可直接参考我们的代码

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/14.SVD/svdRecommend.py

要点补充

> 基于内容(content-based)的推荐

1. 通过各种标签来标记菜肴
2. 将这些属性作为相似度计算所需要的数据
3. 这就是：基于内容的推荐。

> 构建推荐引擎面临的挑战

问题

- 1）在大规模的数据集上，SVD分解会降低程序的速度
- 2）存在其他很多规模扩展性的挑战性问题，比如矩阵的表示方法和计算相似度得分消耗资源。
- 3）如何在缺乏数据时给出好的推荐-称为冷启动【简单说：用户不会喜欢一个无效的物品，而用户不喜欢的物品又无效】

建议

- 1）在大型系统中，SVD分解(可以在程序调入时运行一次)每天运行一次或者其频率更低，并且还要离线运行。
- 2）在实际中，另一个普遍的做法就是离线计算并保存相似度得分。(物品相似度可能被用户重复的调用)
- 3）冷启动问题，解决方案就是将推荐看成是搜索问题，通过各种标签 / 属性特征进行`基于内容的推荐`。

项目案例: 基于 SVD 的图像压缩

> 收集 并 准备数据

将文本数据转化为矩阵

```
# 加载并转换数据
def imgLoadData(filename):
    myl = []
    # 打开文本文件，并从文件以数组方式读入字符
```

```
    for line in open(filename).readlines():
        newRow = []
        for i in range(32):
            newRow.append(int(line[i]))
        myl.append(newRow)
    # 矩阵调入后，就可以在屏幕上输出该矩阵
    myMat = mat(myl)
    return myMat
```

> 分析数据: 分析 Sigma 的长度个数

通常保留矩阵 80% ~ 90% 的能量，就可以得到重要的特征并去除噪声。

```
def analyse_data(Sigma, loopNum=20):
    """analyse_data(分析 Sigma 的长度取值)

    Args:
        Sigma         Sigma的值
        loopNum       循环次数
    """
    # 总方差的集合（总能量值）
    Sig2 = Sigma**2
    SigmaSum = sum(Sig2)
    for i in range(loopNum):
        SigmaI = sum(Sig2[:i+1])
        '''
        根据自己的业务情况，就行处理，设置对应的 Singma 次数

        通常保留矩阵 80% ~ 90% 的能量，就可以得到重要的特征并取出噪声。
        '''
        print '主成分：%s，方差占比：%s%%' % (format(i+1, '2.0f'), format(SigmaI/SigmaSum*100, '4.2f'))
```

> 使用算法: 对比使用 SVD 前后的数据差异对比，对于存储大家可以试着写写

例如：`32*32=1024 => 32*2+2*1+32*2=130`(2*1表示去掉了除对角线的0), 几乎获得了10倍的压缩比。

```
# 打印矩阵
def printMat(inMat, thresh=0.8):
    # 由于矩阵保护了浮点数，因此定义浅色和深色，遍历所有矩阵元素，当元素大于阀值时打印1，否则打印0
    for i in range(32):
        for k in range(32):
            if float(inMat[i, k]) > thresh:
                print 1,
            else:
                print 0,
        print ''


# 实现图像压缩，允许基于任意给定的奇异值数目来重构图像
def imgCompress(numSV=3, thresh=0.8):
    """imgCompress( )

    Args:
        numSV       Sigma长度
        thresh      判断的阈值
    """
    # 构建一个列表
    myMat = imgLoadData('input/14.SVD/0_5.txt')

    print "****original matrix****"
    # 对原始图像进行SVD分解并重构图像e
    printMat(myMat, thresh)

    # 通过Sigma 重新构成SigRecom来实现
    # Sigma是一个对角矩阵，因此需要建立一个全0矩阵，然后将前面的那些奇异值填充到对角线上。
    U, Sigma, VT = la.svd(myMat)
    # SigRecon = mat(zeros((numSV, numSV)))
    # for k in range(numSV):
    #     SigRecon[k, k] = Sigma[k]

    # 分析插入的 Sigma 长度
    analyse_data(Sigma, 20)

    SigRecon = mat(eye(numSV) * Sigma[: numSV])
    reconMat = U[:, :numSV] * SigRecon * VT[:numSV, :]
    print "****reconstructed matrix using %d singular values *****" % numSV
    printMat(reconMat, thresh)
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/14.SVD/svdRecommend.py

- 作者：片刻 1988
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

第15章 大数据与MapReduce

大数据 概述

大数据：收集到的数据已经远远超出了我们的处理能力。

大数据 场景

```
假如你为一家网络购物商店工作，很多用户访问该网站，其中有些人会购买商品，有些人则随意浏览后就离开。
对于你来说，可能很想识别那些有购物意愿的用户。
那么问题就来了，数据集可能会非常大，在单机上训练要运行好几天。
接下来：我们讲讲 MapRedece 如何来解决这样的问题
```

MapRedece

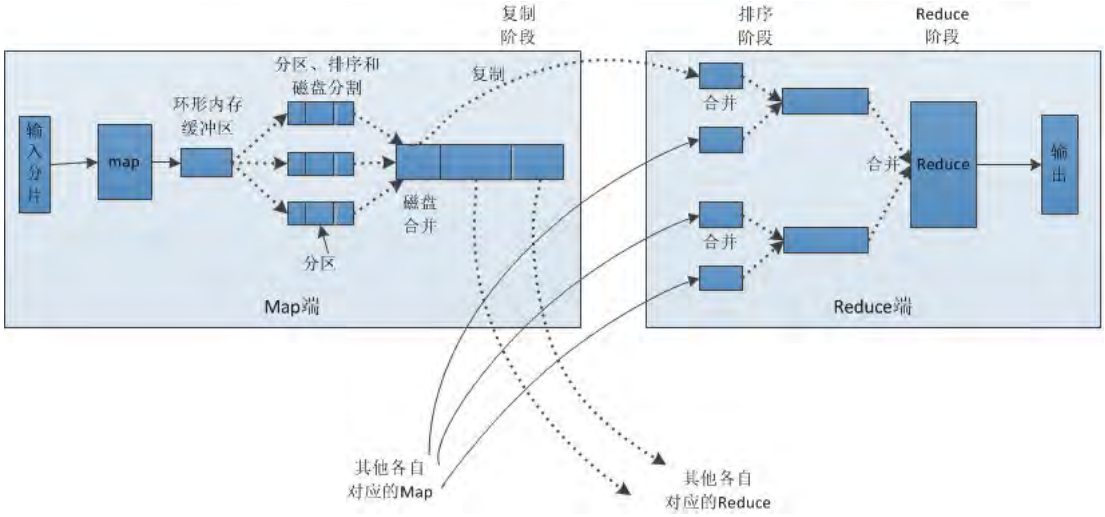
Hadoop 概述

```
Hadoop 是 MapRedece 框架的一个免费开源实现。
MapReduce: 分布式的计算框架，可以将单个计算作业分配给多台计算机执行。
```

MapRedece 原理

> MapRedece 工作原理

- 主节点控制 MapReduce 的作业流程
- MapReduce 的作业可以分成map任务和reduce任务
- map 任务之间不做数据交流，reduce 任务也一样
- 在 map 和 reduce 阶段中间，有一个 sort 和 combine 阶段
- 数据被重复存放在不同的机器上，以防止某个机器失效
- mapper 和 reducer 传输的数据形式为 key/value对



> MapRedece 特点

```
优点：使程序以并行的方式执行，可在短时间内完成大量工作。
缺点：算法必须经过重写，需要对系统工程有一定的理解。
适用数据类型：数值型和标称型数据。
```

Hadoop 流(Python 调用)

> 理论简介

例如: Hadoop流可以像Linux命令一样执行

```
cat inputFile.txt | python mapper.py | sort | python reducer.py > outputFile.txt
```

类似的Hadoop流就可以在多台机器上分布式执行，用户可以通过Linux命令来测试Python语言编写的MapReduce脚本。

> 实战脚本

```
# 测试 Mapper
# Linux
cat input/15.BigData_MapReduce/inputFile.txt | python src/python/15.BigData_MapReduce/mrMeanMapper.py
# Window
# python src/python/15.BigData_MapReduce/mrMeanMapper.py < input/15.BigData_MapReduce/inputFile.txt

# 测试 Reducer
# Linux
cat input/15.BigData_MapReduce/inputFile.txt | python src/python/15.BigData_MapReduce/mrMeanMapper.py | python src/python/15.BigData_MapReduce/m
# Window
# python src/python/15.BigData_MapReduce/mrMeanMapper.py < input/15.BigData_MapReduce/inputFile.txt | python src/python/15.BigData_MapReduce/mrM
```

MapReduce 机器学习

Mahout in Action

1. 简单贝叶斯：它属于为数不多的可以很自然的使用MapReduce的算法。通过统计在某个类别下某特征的概率。
2. k-近邻算法：高维数据下（如文本、图像和视频）流行的近邻查找方法是局部敏感哈希算法。
3. 支持向量机(SVM)：使用随机梯度下降算法求解，如Pegasos算法。
4. 奇异值分解：Lanczos算法是一个有效的求解近似特征值的算法。
5. k-均值聚类：canopy算法初始化k个簇，然后再运行K-均值求解结果。

使用 mrjob 库将 MapReduce 自动化

> 理论简介

- MapReduce 作业流自动化的框架：Cascading 和 Oozie.
- mrjob 是一个不错的学习工具，与2010年底实现了开源，来之于 Yelp(一个餐厅点评网站).

```
python src/python/15.BigData_MapReduce/mrMean.py < input/15.BigData_MapReduce/inputFile.txt > input/15.BigData_MapReduce/myOut.txt
```

> 实战脚本

```
# 测试 mrjob的案例
# 先测试一下mapper方法
# python src/python/15.BigData_MapReduce/mrMean.py --mapper < input/15.BigData_MapReduce/inputFile.txt
# 运行整个程序，移除 --mapper 就行
python src/python/15.BigData_MapReduce/mrMean.py < input/15.BigData_MapReduce/inputFile.txt
```

项目案例：分布式 SVM 的 Pegasos 算法

Pegasos是指原始估计梯度求解器(Peimal Estimated sub-GrAdient Solver)

Pegasos 工作原理

1. 从训练集中随机挑选一些样本点添加到带处理列表中
2. 按序判断每个样本点是否被正确分类
   - 如果是则忽略
   - 如果不是则将其加入到待更新集合。
3. 批处理完毕后，权重向量按照这些错分的样本进行更新。

上述算法伪代码如下：

```
将 回归系数w 初始化为0
对每次批处理
    随机选择 k 个样本点(向量)
    对每个向量
        如果该向量被错分：
            更新权重向量 w
    累加对 w 的更新
```

开发流程

```
收集数据：数据按文本格式存放。
准备数据：输入数据已经是可用的格式，所以不需任何准备工作。如果你需要解析一个大规模的数据集，建议使用 map 作业来完成，从而达到并行处理的目的。
分析数据：无。
训练算法：与普通的 SVM 一样，在分类器训练上仍需花费大量的时间。
测试算法：在二维空间上可视化之后，观察超平面，判断算法是否有效。
使用算法：本例不会展示一个完整的应用，但会展示如何在大数据集上训练SVM。该算法其中一个应用场景就是本文分类，通常在文本分类里可能有大量的文档和成千上万的特征。
```

> 收集数据

文本文件数据格式如下：

```
0.365032        2.465645        -1
-2.494175       -0.292380       -1
-3.039364       -0.123108       -1
1.348150        0.255696        1
2.768494        1.234954        1
1.232328        -0.601198       1
```

> 准备数据

```
def loadDataSet(fileName):
    dataMat = []
    labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        # dataMat.append([float(lineArr[0]), float(lineArr[1]), float(lineArr[2])])
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat, labelMat
```

> 分析数据: 无

> 训练算法

```
def batchPegasos(dataSet, labels, lam, T, k):
    """batchPegasos()

    Args:
        dataMat    特征集合
        labels     分类结果集合
        lam        固定值
        T          迭代次数
        k          待处理列表大小
    Returns:
        w          回归系数
    """
    m, n = shape(dataSet)
    w = zeros(n)  # 回归系数
    dataIndex = range(m)
    for t in range(1, T+1):
        wDelta = mat(zeros(n))  # 重置 wDelta

        # 它是学习率，代表了权重调整幅度的大小。（也可以理解为随机梯度的步长，使它不断减小，便于拟合）
```

```
        # 输入T和K分别设定了迭代次数和待处理列表的大小。在T次迭代过程中，每次需要重新计算eta
        eta = 1.0/(lam*t)
        random.shuffle(dataIndex)
        for j in range(k):      # 全部的训练集  内循环中执行批处理，将分类错误的值全部做累加后更新权重向量
            i = dataIndex[j]
            p = predict(w, dataSet[i, :])              # mapper 代码

            # 如果预测正确，并且预测结果的绝对值>=1，因为最大间隔为1，认为没问题。
            # 否则算是预测错误，通过预测错误的结果，来累计更新w.
            if labels[i]*p < 1:                        # mapper 代码
                wDelta += labels[i]*dataSet[i, :].A    # 累积变化
        # w通过不断的随机梯度的方式来优化
        w = (1.0 - 1/t)*w + (eta/k)*wDelta             # 在每个 T上应用更改
        # print '-----', w
    # print '++++++', w
    return w
```

完整代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/15.BigData_MapReduce/pegasos.py

MR版本的代码地址: https://github.com/apachecn/MachineLearning/blob/master/src/python/15.BigData_MapReduce/mrSVM.py

---

- 作者：片刻 小瑶
- GitHub地址: https://github.com/apachecn/MachineLearning

第16章 推荐系统

背景与挖掘目标

随着互联网的快速发展，用户很难快速从海量信息中寻找到自己感兴趣的信息。因此诞生了：搜索引擎+推荐系统

本章节-推荐系统：

1. 帮助用户发现其感兴趣和可能感兴趣的信息。
2. 让网站价值信息脱颖而出，得到广大用户的认可。
3. 提高用户对网站的忠诚度和关注度，建立稳固用户群体。

分析方法与过程

本案例的目标是对用户进行推荐，即以一定的方式将用户与物品（本次指网页）之间建立联系。

由于用户访问网站的数据记录很多，如果不对数据进行分类处理，对所有的记录直接采用推荐系统进行推荐，这样会存在一下问题。

1. 数据量太大意味着物品数与用户数很多，在模型构建用户与物品稀疏矩阵时，出现设备内存空间不够的情况，并且模型计算需要消耗大量的时间。
2. 用户区别很大，不同的用户关注的信息不一样，因此，即使能够得到推荐结果，其效果也会不好。

为了避免出现上述问题，需要进行分类处理与分析。

正常的情况下，需要对用户的兴趣爱好以及需求进行分类。 因为在用户访问记录中，没有记录用户访问页面时间的长短，因此不容易判断用户兴趣爱好。 因此，本文根据用户浏览的网页信息进行分析处理，主要采用以下方法处理：以用户浏览网页的类型进行分类，然后对每个类型中的内容进行推荐。

分析过程如下：

- 从系统中获取用户访问网站的原始记录。
- 对数据进行多维分析，包括用户访问内容，流失用户分析以及用户分类等分析。
- 对数据进行预处理，包含数据去重、数据变换和数据分类鞲处理过程。
- 以用户访问html后缀的页面为关键条件，对数据进行处理。
- 对比多种推荐算法进行推荐，通过模型评价，得到比较好的智能推荐模型。通过模型对样本数据进行预测，获得推荐结果。

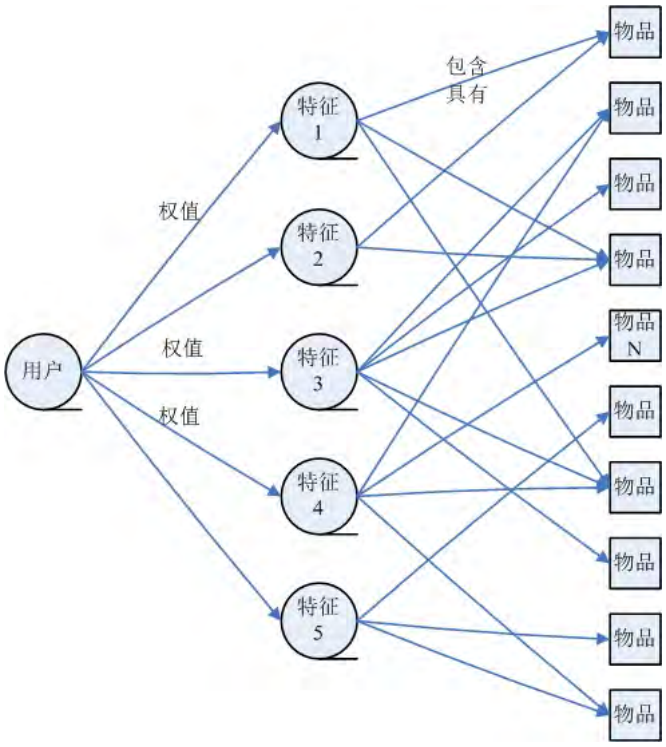
主流推荐算法

推荐方法	描述
基于内容推荐	
协同过滤推荐	
基于规则推荐	
基于效用推荐	
基于知识推荐	
组合推荐	

推荐方法	优点	缺点
基于内容推荐	推荐结果直观，容易解释； 不需要领域知识	稀疏问题；新用户问题； 复杂属性不好处理； 要有足够数据构造分类器
协同过滤推荐	新异兴趣发现、不需要领域知 随着时间推移性能提高； 推荐个性化、自动化程度高； 能处理复杂的非结构化对象	稀疏问题； 可扩展性问题； 新用户问题； 质量取决于历史数据集； 系统开始时推荐质量差；
基于规则推荐	能发现新兴趣点； 不要领域知识	规则抽取难、耗时； 产品名同义性问题； 个性化程度低；
基于效用推荐	无冷开始和稀疏问题； 对用户偏好变化敏感； 能考虑非产品特性	用户必须输入效用函数； 推荐是静态的，灵活性差； 属性重叠问题；
基于知识推荐	能把用户需求映射到产品上； 能考虑非产品属性	知识难获得； 推荐是静态的

基于知识推荐

基于知识的推荐（Knowledge-based Recommendation）在某种程度是可以看成是一种推理（Inference）技术，它不是建立在用户需要和偏好基础上推荐的。基于知识的方法因它们所用的功能知识不同而有明显区别。效用知识（Functional Knowledge）是一种关于一个项目如何满足某一特定用户的知识，因此能解释需要和推荐的关系，所以用户资料可以是任何能支持推理的知识结构，它可以是用户已经规范化的查询，也可以是一个更详细的用户需要的表示。



协同过滤推荐

- memory-based推荐
  - Item-based方法
  - User-based方法
  - Memory-based推荐方法通过执行最近邻搜索，把每一个Item或者User看成一个向量，计算其他所有Item或者User与它的相似度。有了Item或者User之间的两两相似度之后，就可以进行预测与推荐了。
- model-based推荐
  - Model-based推荐最常见的方法为Matrix factorization.
  - 矩阵分解通过把原始的评分矩阵R分解为两个矩阵相乘，并且只考虑有评分的值，训练时不考虑missing项的值。R矩阵分解成为U与V两个矩阵后，评分矩阵R中missing的值就可以通过U矩阵中的某列和V矩阵的某行相乘得到
  - 矩阵分解的目标函数: U矩阵与V矩阵的可以通过梯度下降(gradient descent)算法求得，通过交替更新u与v多次迭代收敛之后可求出U与V。
  - 矩阵分解背后的核心思想，找到两个矩阵，它们相乘之后得到的那个矩阵的值，与评分矩阵R中有值的位置中的值尽可能接近。这样一来，分解出来的两个矩阵相乘就尽可能还原了评分矩阵R，因为有值的地方，值都相差得尽可能地小，那么missing的值通过这样的方式计算得到，比较符合趋势。
- 协同过滤中主要存在如下两个问题：稀疏性与冷启动问题。已有的方案通常会通过引入多个不同的数据源或者辅助信息(Side information)来解决这些问题，用户的Side information可以是用户的基本个人信息、用户画像信息等，而Item的Side information可以是物品的content信息等。

效果评估

1. 召回率和准确率 【人为统计分析】

2. F值(P-R曲线)【偏重：非均衡问题】
3. ROC和AUC【偏重：不同结果的对比】

- 作者：片刻
- GitHub地址: https://github.com/apachecn/MachineLearning
- 版权声明：欢迎转载学习 => 请标注信息来源于 ApacheCN

> 摘录的原文地址：

- 推荐系统中常用算法 以及优点缺点对比
- 推荐算法的基于知识推荐
- 推荐系统中基于深度学习的混合协同过滤模型