第1章 机器学习基础

机器学习 概述

`机器学习(Machine Learning,ML)` 是使用计算机来彰显数据背后的真实含义，它为了把无序的数据转换成有用的信息。是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。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的性能。
它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎。

1. 海量的数据
2. 获取有用的信息

机器学习 研究意义

机器学习是一门人工智能的科学，该领域的主要研究对象是人工智能，特别是如何在经验学习中改善具体算法的性能”。 “机器学习是对能通过经验自动改进的计算机算法的研究”。 “机器学习是用数据或以往的经验，以此优化计算机程序的性能标准。” 一种经常引用的英文定义是: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

机器学习已经有了十分广泛的应用，例如: 数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA序列测序、语音和手写识别、战略游戏和机器人运用。

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:
    - **深度学习 vs 机器学习 vs 模式识别**
    - **深度学习 百科资料**

> 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,是机器学习来帮助搜索引擎判断哪个结果更适合你（也判断哪个广告更适合你）。
- 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（classification）: 将实例数据划分到合适的类别中。
    - 应用实例: 判断网站是否被黑客入侵（二分类），手写数字的自动识别（多分类）
- 回归（regression）: 主要用于预测数值型数据。
    - 应用实例: 股票价格波动的预测，房屋价格的预测等。

监督学习（supervised learning）

- 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。在监督学习中，给定一组数据，我们知道正确的输出结果应该是什么样子，并且知道在输入和输出之间有着一个特定的关系。（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)。
- 监督学习需要注意的问题:
    - 偏置方差权衡
    - 功能的复杂性和数量的训练数据
    - 输入空间的维数
    - 噪声中的输出值
- 知识表示:

- 可以采用规则集的形式【例如: 数学成绩大于90分为优秀】
- 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】
- 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】

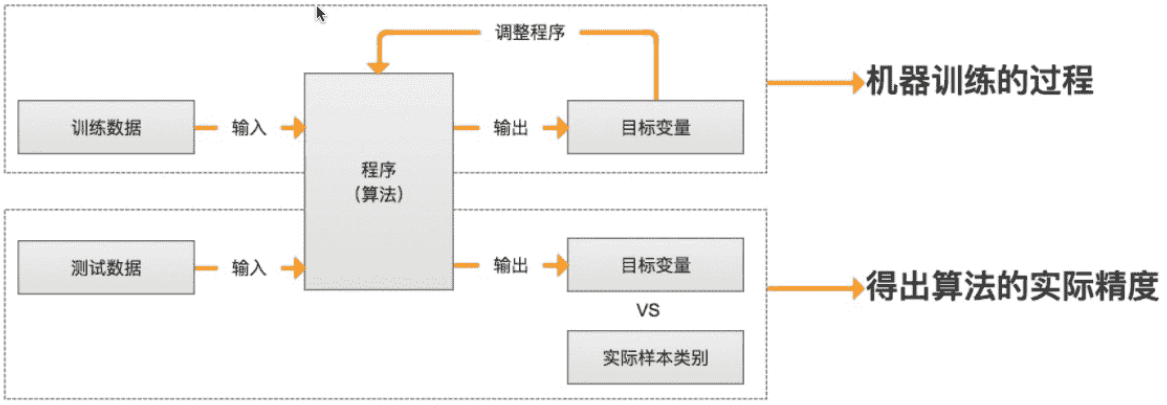
非监督学习（unsupervised learning）

- 在机器学习，无监督学习的问题是，在未加标签的数据中，试图找到隐藏的结构。因为提供给学习者的实例是未标记的，因此没有错误或报酬信号来评估潜在的解决方案。
- 无监督学习是密切相关的统计数据密度估计的问题。然而无监督学习还包括寻求，总结和解释数据的主要特点等诸多技术。在无监督学习使用的许多方法是基于用于处理数据的数据挖掘方法。
- 数据没有类别信息，也不会给定目标值。
- 非监督学习包括的类型:
  - 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: 通过样本分布的紧密程度，来估计与分组的相似性。
  - 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。

强化学习

这个算法可以训练程序做出某一决定。程序在某一情况下尝试所有的可能行动，记录不同行动的结果并试着找出最好的一次尝试来做决定。 属于这一类算法的有马尔可夫决策过程。

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

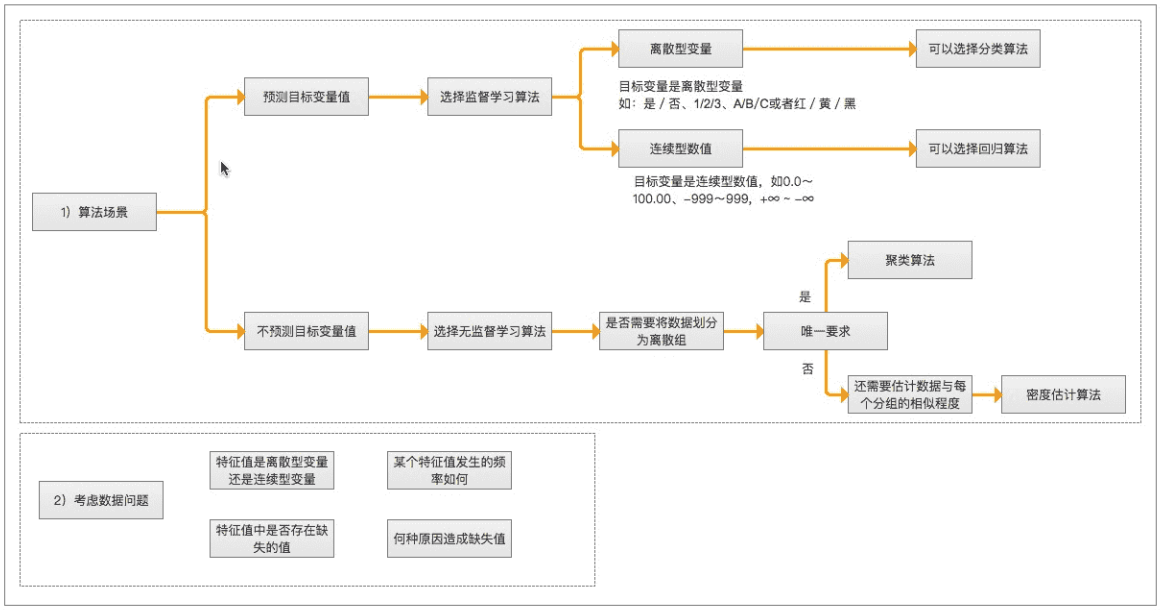
监督学习的用途	
k-近邻算法	线性回归
朴素贝叶斯算法	局部加权线性回归
支持向量机	Ridge 回归
决策树	Lasso 最小回归系数估计
无监督学习的用途	
K-均值	最大期望算法
DBSCAN	Parzen窗设计

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

1. 收集数据: 收集样本数据
2. 准备数据: 注意数据的格式
3. 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。
4. 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
5. 测试算法: [机器学习算法核心]评估算法效果
6. 使用算法: 将机器学习算法转为应用程序

机器学习 数学基础

- 微积分
- 统计学/概率论
- 线性代数

机器学习 工具

Python语言

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`
    - 数据分析库 `Pandas`

数学工具

- Matlab

附: 机器学习专业术语

- 模型（model）: 计算机层面的认知
- 学习算法（learning algorithm），从数据中产生模型的方法
- 数据集（data set）: 一组记录的合集
- 示例（instance）: 对于某个对象的描述
- 样本（sample）: 也叫示例
- 属性（attribute）: 对象的某方面表现或特征
- 特征（feature）: 同属性
- 属性值（attribute value）: 属性上的取值
- 属性空间（attribute space）: 属性张成的空间
- 样本空间/输入空间（samplespace）: 同属性空间
- 特征向量（feature vector）: 在属性空间里每个点对应一个坐标向量，把一个示例称作特征向量
- 维数（dimensionality）: 描述样本参数的个数（也就是空间是几维的）
- 学习（learning）/训练（training）: 从数据中学得模型
- 训练数据（training data）: 训练过程中用到的数据
- 训练样本（training sample）:训练用到的每个样本
- 训练集（training set）: 训练样本组成的集合
- 假设（hypothesis）: 学习模型对应了关于数据的某种潜在规则
- 真相（ground-truth）:真正存在的潜在规律
- 学习器（learner）: 模型的另一种叫法，把学习算法在给定数据和参数空间的实例化
- 预测（prediction）: 判断一个东西的属性
- 标记（label）: 关于示例的结果信息，比如我是一个“好人”。
- 样例（example）: 拥有标记的示例
- 标记空间/输出空间（label space）: 所有标记的集合
- 分类（classification）: 预测是离散值，比如把人分为好人和坏人之类的学习任务
- 回归（regression）: 预测值是连续值，比如你的好人程度达到了0.9，0.6之类的
- 二分类（binary classification）: 只涉及两个类别的分类任务
- 正类（positive class）: 二分类里的一个
- 反类（negative class）: 二分类里的另外一个
- 多分类（multi-class classification）: 涉及多个类别的分类

- 测试（testing）: 学习到模型之后对样本进行预测的过程
- 测试样本（testing sample）: 被预测的样本
- 聚类（clustering）: 把训练集中的对象分为若干组
- 簇（cluster）: 每一个组叫簇
- 监督学习（supervised learning）: 典范--分类和回归
- 无监督学习（unsupervised learning）: 典范--聚类
- 未见示例（unseen instance）: “新样本“，没训练过的样本
- 泛化（generalization）能力: 学得的模型适用于新样本的能力
- 分布（distribution）: 样本空间的全体样本服从的一种规律
- 独立同分布（independent and identically distributed，简称i,i,d.）:获得的每个样本都是独立地从这个分布上采样获得的。

机器学习基础补充

数据集的划分

- 训练集（Training set） —— 学习样本数据集，通过匹配一些参数来建立一个模型，主要用来训练模型。类比考研前做的解题大全。
- 验证集（validation set） —— 对学习出来的模型，调整模型的参数，如在神经网络中选择隐藏单元数。验证集还用来确定网络结构或者控制模型复杂程度的参数。类比 考研之前做的模拟考试。
- 测试集（Test set） —— 测试训练好的模型的分辨能力。类比 考研。这次真的是一考定终身。

模型拟合程度

- 欠拟合（Underfitting）: 模型没有很好地捕捉到数据特征，不能够很好地拟合数据，对训练样本的一般性质尚未学好。类比，光看书不做题觉得自己什么都会了，上了考场才知道自己啥都不会。
- 过拟合（Overfitting）: 模型把训练样本学习“太好了”，可能把一些训练样本自身的特性当做了所有潜在样本都有的一般性质，导致泛化能力下降。类比，做课后题全都做对了，超纲题也都认为是考试必考题目，上了考场还是啥都不会。

通俗来说，欠拟合和过拟合都可以用一句话来说，欠拟合就是: “你太天真了！”，过拟合就是: “你想太多了！”。

常见的模型指标

- 正确率 —— 提取出的正确信息条数 / 提取出的信息条数
- 召回率 —— 提取出的正确信息条数 / 样本中的信息条数
- F 值 —— 正确率 * 召回率 * 2 / （正确率 + 召回率）（F值即为正确率和召回率的调和平均值）

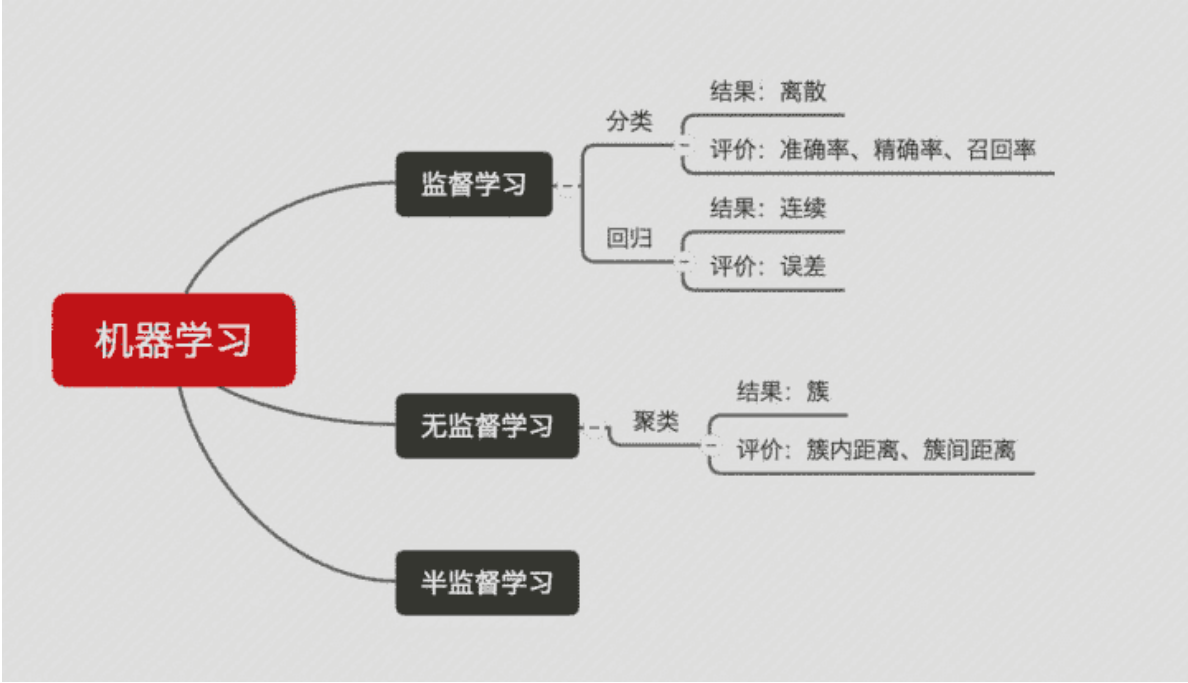
举个例子如下:

举个例子如下:
某池塘有 1400 条鲤鱼，300 只虾，300 只乌龟。现在以捕鲤鱼为目的。撒了一张网，逮住了 700 条鲤鱼，200 只
虾， 100 只乌龟。那么这些指标分别如下:
正确率 = 700 / (700 + 200 + 100) = 70%
召回率 = 700 / 1400 = 50%
F 值 = 70% * 50% * 2 / (70% + 50%) = 58.3%

模型

- 分类问题 —— 说白了就是将一些未知类别的数据分到现在已知的类别中去。比如，根据你的一些信息，判断你是高富帅，还是穷屌丝。评判分类效果好坏的三个指标就是上面介绍的三个指标: 正确率，召回率，F值。
- 回归问题 —— 对数值型连续随机变量进行预测和建模的监督学习算法。回归往往会通过计算 误差（Error）来确定模型的精确性。
- 聚类问题 —— 聚类是一种无监督学习任务，该算法基于数据的内部结构寻找观察样本的自然族群（即集群）。聚类问题的标准一般基于距离: 簇内距离（Intra-cluster Distance） 和 簇间距离（Inter-cluster Distance） 。簇内距离是越小越好，也就是簇内的元素越相似越好；而簇间距离越大越好，也就是说簇间（不同簇）元素越不相同越好。一般的，衡量聚类问题会给出一个结合簇内距离和簇间距离的公式。

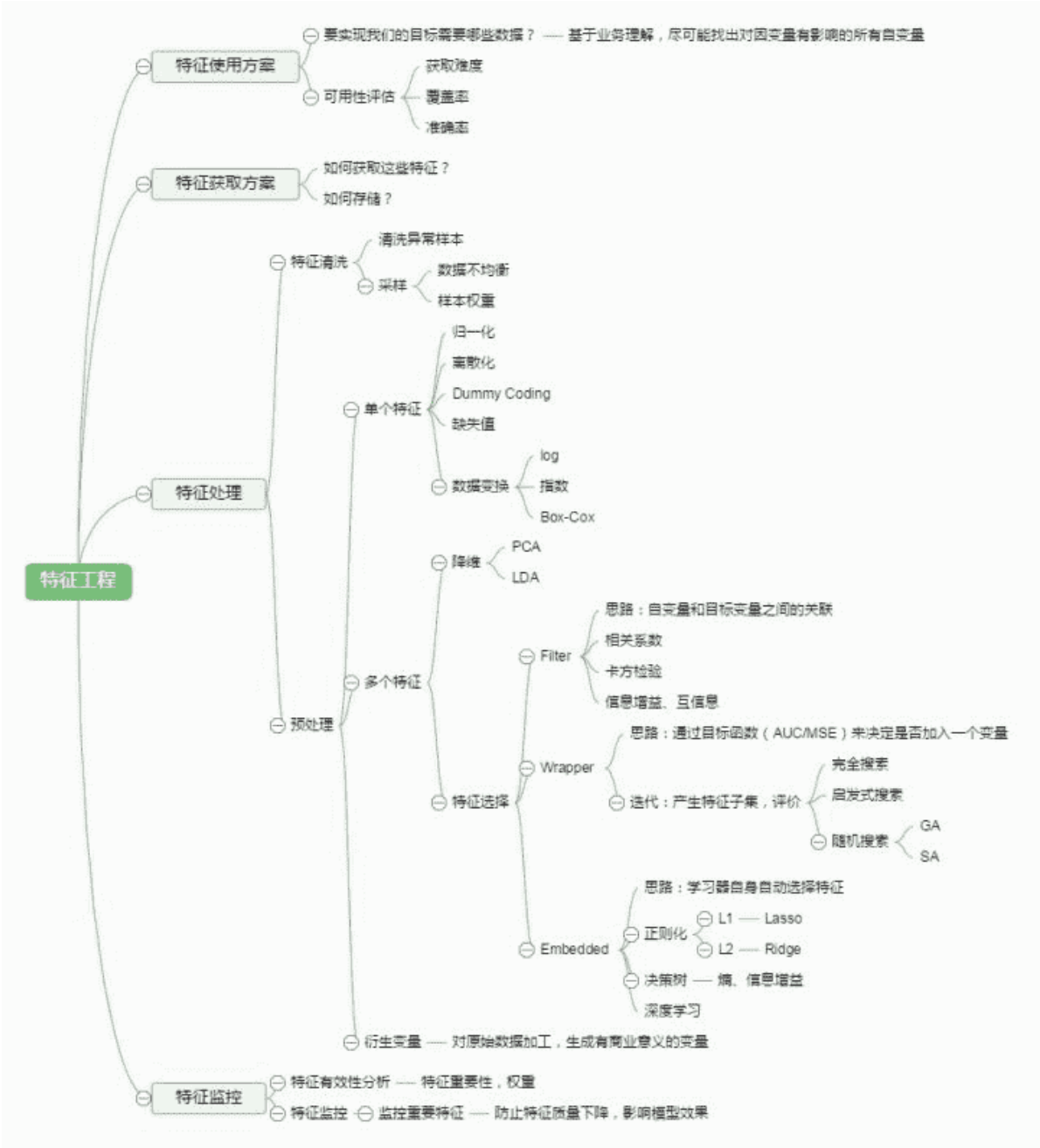
下面这个图可以比较直观地展示出来:



特征工程的一些小东西

- 特征选择 —— 也叫特征子集选择（FSS，Feature Subset Selection）。是指从已有的 M 个特征（Feature）中选择 N 个特征使得系统的特定指标最优化，是从原始特征中选择出一些最有效特征以降低数据集维度的过程，是提高算法性能的一个重要手段，也是模式识别中关键的数据预处理步骤。
- 特征提取 —— 特征提取是计算机视觉和图像处理中的一个概念。它指的是使用计算机提取图像信息，决定每个图像的点是否属于一个图像特征。特征提取的结果是把图像上的点分为不同的子集，这些子集往往属于孤立的点，连续的曲线或者连续的区域。

下面给出一个特征工程的图:



其他

- Learning rate —— 学习率，通俗地理解，可以理解为步长，步子大了，很容易错过最佳结果。就是本来目标尽在咫尺，可是因为我迈的步子很大，却一下子走过了。步子小了呢，就是同样的距离，我却要走很多很多步，这样导致训练的耗时费力还不讨好。

第2章 k-近邻算法

KNN 概述

k-近邻（kNN, k-NearestNeighbor）算法是一种基本分类与回归方法，我们这里只讨论分类问题中的 k-近邻算法。

一句话总结: 近朱者赤近墨者黑!

k 近邻算法的输入为实例的特征向量，对应于特征空间的点；输出为实例的类别，可以取多类。k 近邻算法假设给定一个训练数据集，其中的实例类别已定。分类时，对新的实例，根据其 k 个最近邻的训练实例的类别，通过多数表决等方式进行预测。因此，k近邻算法不具有显式的学习过程。

k 近邻算法实际上利用训练数据集对特征向量空间进行划分，并作为其分类的“模型”。 k值的选择、距离度量以及分类决策规则是k近邻算法的三个基本要素。

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

1 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。
2 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。
3 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 通俗理解

给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的 k 个实例，这 k 个实例的多数属于某个类，就把该输入实例分为这个类。

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 小结

KNN 是什么？定义：监督学习？ 非监督学习？

KNN 是一个简单的无显示学习过程，非泛化学习的 `监督学习` 模型。在分类和回归中均有应用。

基本原理

简单来说：通过距离度量来计算查询点（query point）与每个训练数据点的距离，然后选出与查询点（query point）相近的K个最邻点（K nearest neighbors），使用分类决策来选出对应的标签来作为该查询点的标签。

KNN 三要素

1.K的取值

对查询点标签影响显著（效果拔群）。k值小的时候 近似误差小，估计误差大。 k值大近似误差大，估计误差小。

如果选择较小的 k 值，就相当于用较小的邻域中的训练实例进行预测，“学习”的近似误差（approximation error）会减小，只有与输入实例较近的（相似的）训练实例才会对预测结果起作用。但缺点是“学习”的估计误差（estimation error）会增大，预测结果会对近邻的实例点非常敏感。如果邻近的实例点恰巧是噪声，预测就会出错。换句话说，k 值的减小就意味着整体模型变得复杂，容易发生过拟合。

如果选择较大的 k 值，就相当于用较大的邻域中的训练实例进行预测。其优点是可以减少学习的估计误差。但缺点是学习的近似误差会增大。这时与输入实例较远的（不相似的）训练实例也会对预测起作用，使预测发生错误。 k 值的增大就意味着整体的模型变得简单。

太大太小都不太好，可以用交叉验证（cross validation）来选取适合的k值。

2.距离度量 Metric/Distance Measure

距离度量 通常为 欧式距离（Euclidean distance），还可以是 Minkowski 距离 或者 曼哈顿距离。也可以是 地理空间中的一些距离公式。（更多细节可以参看 sklearn 中 valid_metric 部分）

3.分类决策 （decision rule）

分类决策 在 分类问题中 通常为通过少数服从多数 来选取票数最多的标签，在回归问题中通常为 K个最邻点的标签的平均值。

第3章 决策树

决策树 概述

决策树（Decision Tree）算法是一种基本的分类与回归方法，是最经常使用的数据挖掘算法之一。我们这章节只讨论用于分类的决策树。

决策树模型呈树形结构，在分类问题中，表示基于特征对实例进行分类的过程。它可以认为是 if-then 规则的集合，也可以认为是定义在特征空间与类空间上的条件概率分布。

决策树学习通常包括 3 个步骤：特征选择、决策树的生成和决策树的修剪。

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:

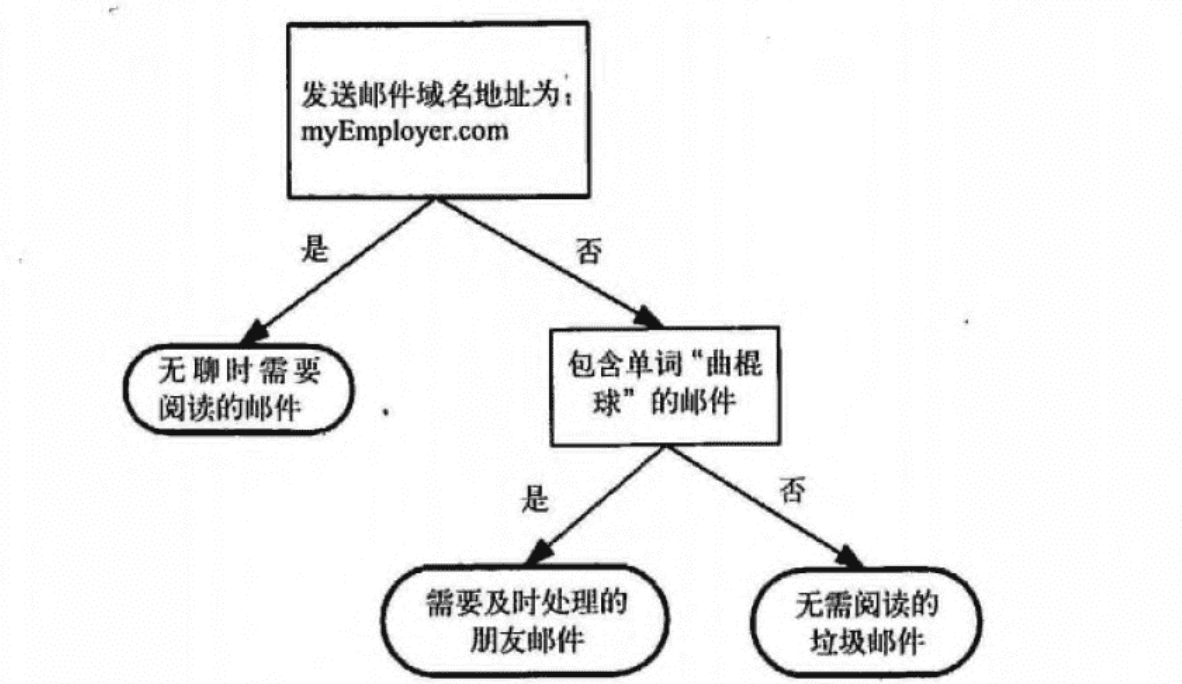


图3-1　流程图形式的决策树

```
1  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。
2  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",
3  如果不包含则将邮件归类到 "无需阅读的垃圾邮件"。
```

决策树的定义:

分类决策树模型是一种描述对实例进行分类的树形结构。决策树由结点（node）和有向边（directed edge）组成。结点有两种类型: 内部结点（internal node）和叶结点（leaf node）。内部结点表示一个特征或属性(features)，叶结点表示一个类(labels)。

用决策树对需要测试的实例进行分类: 从根节点开始，对实例的某一特征进行测试，根据测试结果，将实例分配到其子结点；这时，每一个子结点对应着该特征的一个取值。如此递归地对实例进行测试并分配，直至达到叶结点。最后将实例分配到叶结点的类中。

决策树 原理

决策树 须知概念

信息熵 & 信息增益

熵（entropy）:
熵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。

信息论（information theory）中的熵（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（information gain）:
在划分数据集前后信息发生的变化称为信息增益。

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（这里使用的是ID3算法，只适用于标称型数据，这就是为什么数值型数据必须离散化。还有其他的树构造算法，比如CART）
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：使用训练好的树计算错误率。
使用算法：此步骤可以适用于任何监督学习任务，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，数据有缺失也能跑，可以处理不相关特征。
缺点：容易过拟合。
适用数据类型：数值型和标称型。
```

第4章 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单的一种：朴素贝叶斯分类。

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:

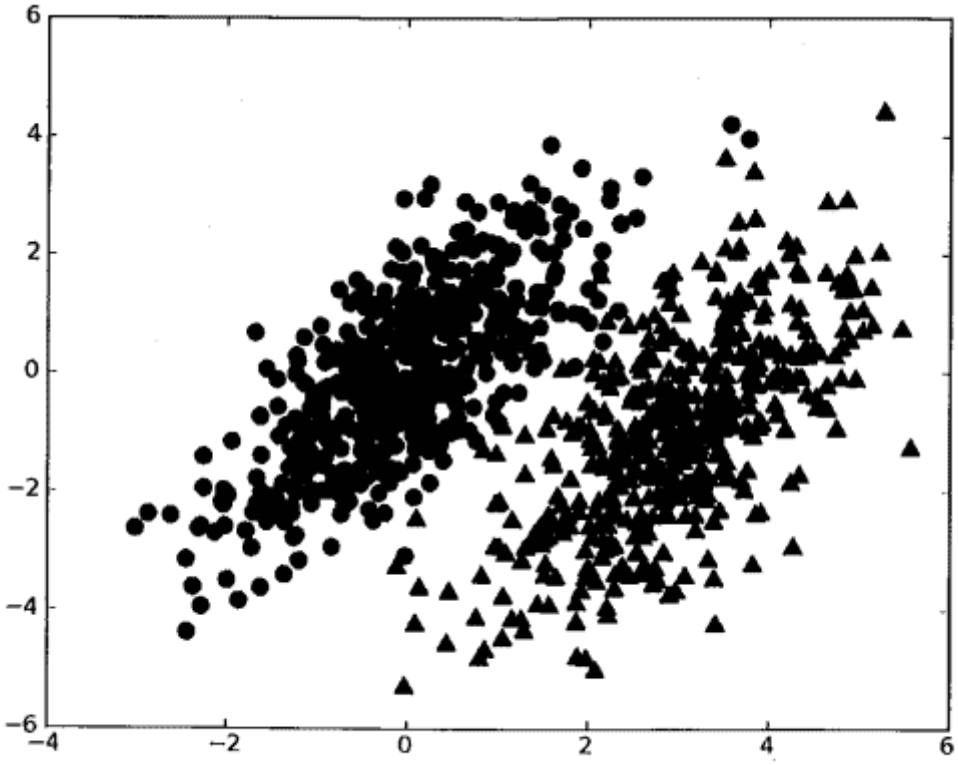


图4-1　两个参数已知的概率分布，参数决定了分布的形状

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。

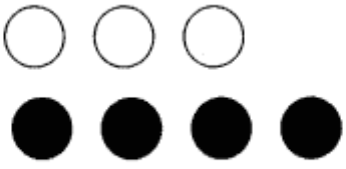


图4-2　一个包含7块石头的集合，石头的颜色为灰色或者黑色。如果随机从中取一块石头，那么取到灰色石头的概率为3/7。类似地，取到黑色石头的概率为4/7

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?

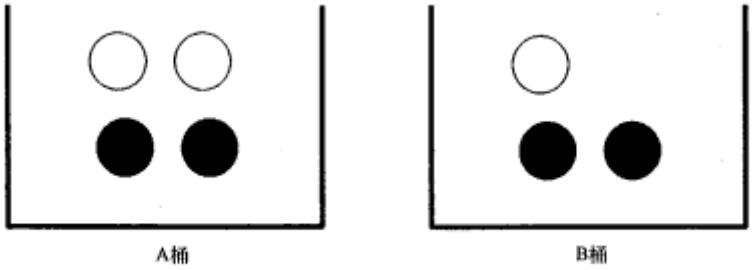


图4-3　落到两个桶中的7块石头

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。
```

第5章 Logistic回归

Logistic 回归 概述

Logistic 回归 或者叫逻辑回归 虽然名字有回归，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(Decision Boundary)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。

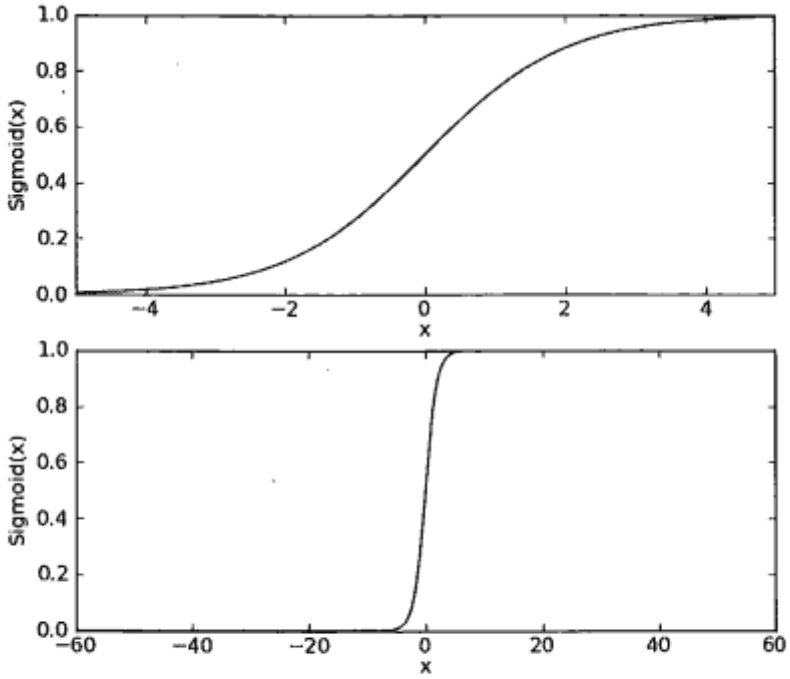


图5-1　两种坐标尺度下的Sigmoid函数图。上图的横坐标为–5到5，这时的曲线变化较为平滑；下图横坐标的尺度足够大，可以看到，在x＝0点处Sigmoid函数看起来很像阶跃函数

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 回归也是一种概率估计，比如这里 Sigmoid 函数得出的值为0.5，可以理解为给定数据和参数，数据被分入 1 类的概率为0.5。想对Sigmoid 函数有更多了解，可以点开 **此链接** 跟此函数互动。

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:

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

如果采用向量的写法，上述公式可以写成 $z = w^T x$ ，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（Gradient Ascent）。

梯度上升法

梯度的介绍

需要一点点向量方面的数学知识

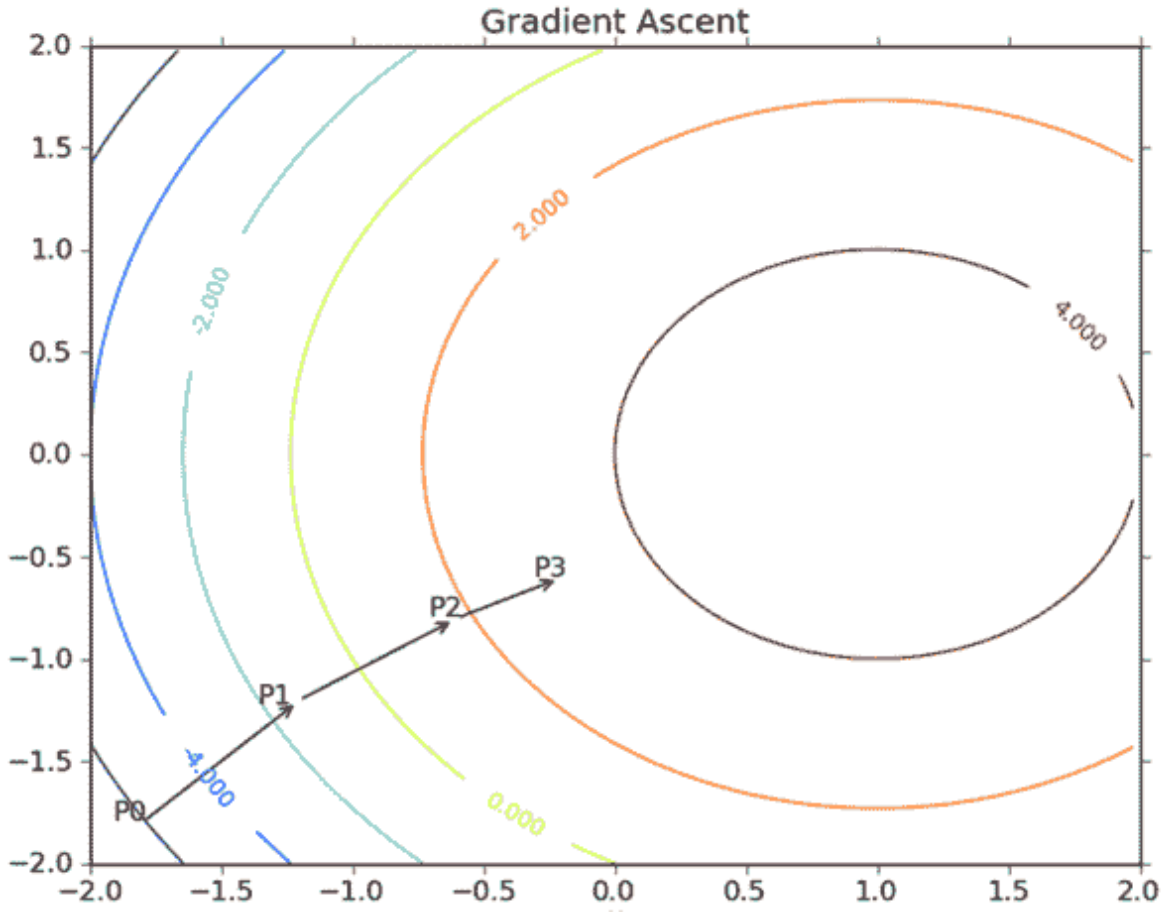
```
向量 = 值 + 方向
梯度 = 向量
梯度 = 梯度值 + 梯度方向
```

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:

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

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 = w0 + 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（step length）。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。
```

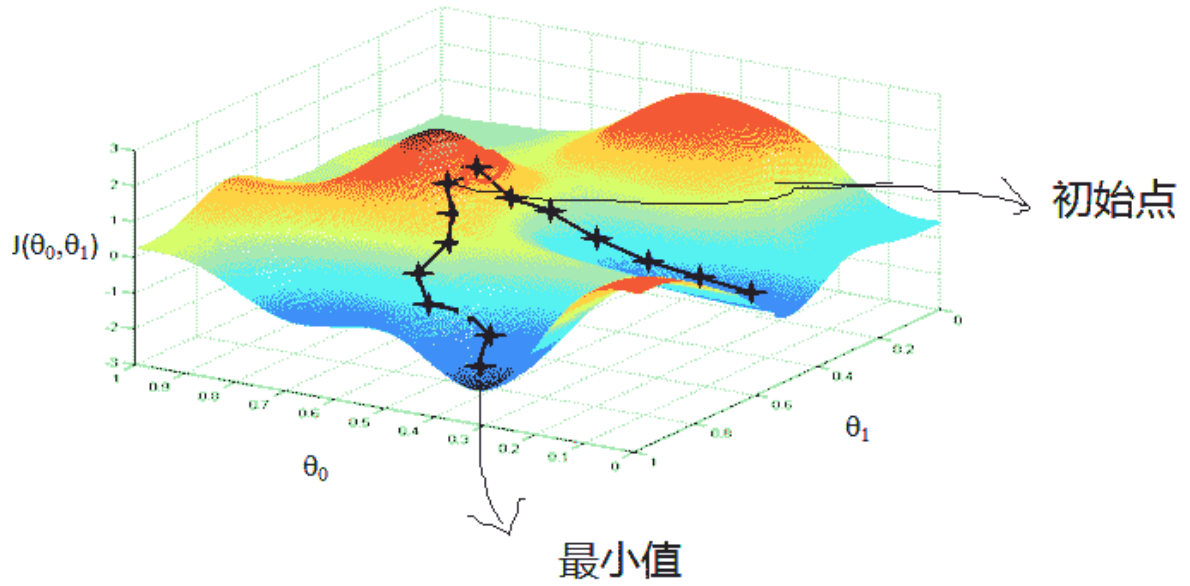
问: 有人会好奇为什么有些书籍上说的是梯度下降法（Gradient Decent）？

答: 其实这个两个方法在此情况下本质上是相同的。关键在于代价函数（cost function）或者叫目标函数（objective function）。如果目标函数是损失函数，那就是最小化损失函数来求函数的最小值，就用梯度下降。 如果目标函数是似然函数（Likelihood function），就是要最大化似然函数来求函数的最大值，那就用梯度上升。在逻辑回归中， 损失函数和似然函数无非就是互为正负关系。

只需要在迭代公式中的加法变成减法。因此，对应的公式可以写成

$$w := w - \alpha \nabla_w f(w)$$

局部最优现象 （Local Optima）



上图表示参数 θ 与误差函数 J(θ) 的关系图 (这里的误差函数是损失函数，所以我们要最小化损失函数)，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（此处的θ0，θ1是x0和x1的系数，也对应的是上文w0和w1）。

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；在这之后，我们就可以在输出的类别上做一些其他分析工作。
```

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。
```

第6章 支持向量机

支持向量机 概述

支持向量机(Support Vector Machines, SVM): 是一种监督学习算法。

- 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。

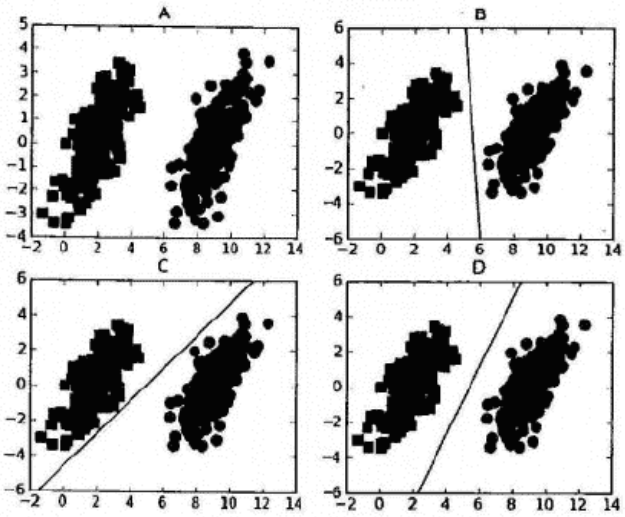
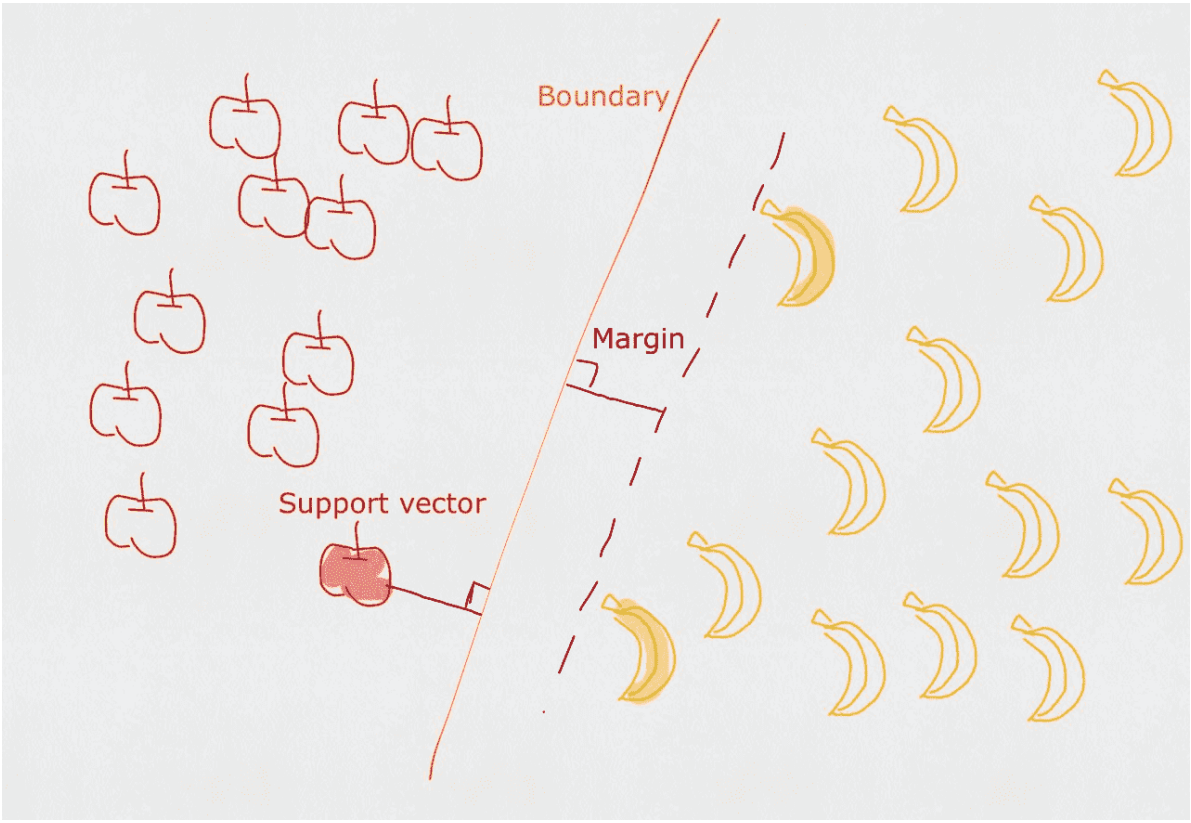


图6-2　A框中给出了一个线性可分的数据集，B、C、D框中各自给出了一条可以将两类数据分开的直线

支持向量机 原理

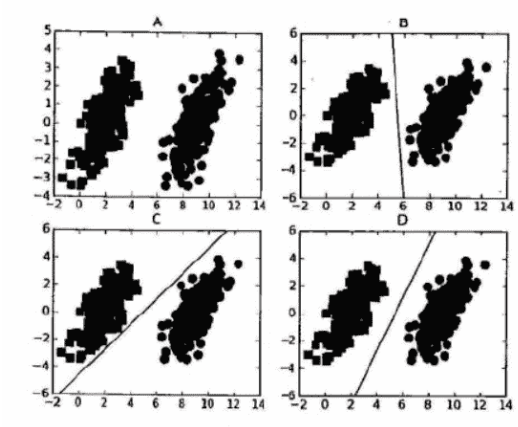
SVM 工作原理



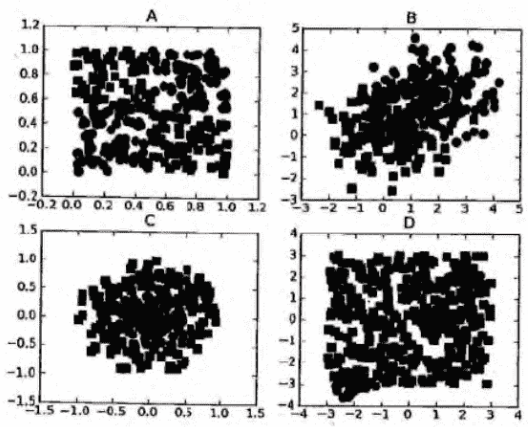
对于上述的苹果和香蕉，我们想象为2种水果类型的炸弹。（保证距离最近的炸弹，距离它们最远）

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?

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 causing a misclassification.
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. 直觉上是最安全的
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），这给我们最小的可能导致错误分类。
3. CV（cross validation 交叉验证）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. 有一些理论表明这是一件好东西。
5. 从经验角度上说它的效果非常非常好。
```

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

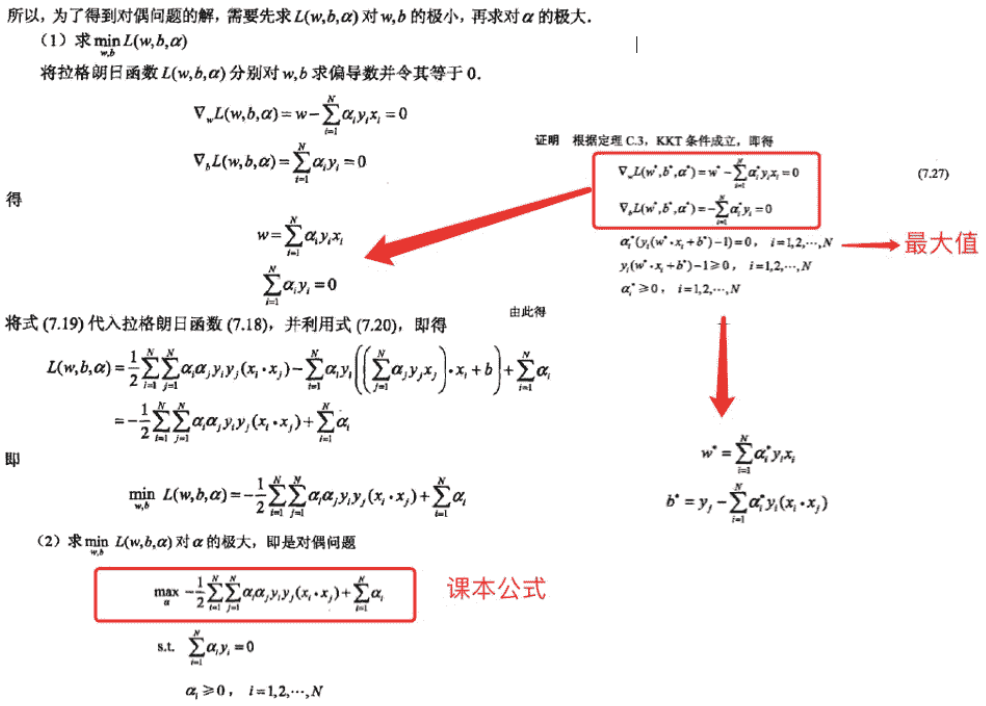
点到直线的距离公式

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

> 拉格朗日乘子法

- 类别标签用-1、1，是为了后期方便 $$label*(w^Tx+b)$$ 的标识和距离计算；如果 $$label*(w^Tx+b)>0$$ 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: $$arg: max_{w, b} \left(min[label*(w^Tx+b)] \frac{1}{||w||} \right) $$
        1. 如果 $$label*(w^Tx+b)>0$$ 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 $$label*(w^Tx+b)>=1$$，因为*0 ~ 1*之间，得到的点是存在误判的可能性，所以要保障 $$min[label*(w^Tx+b)]=1$$，才能更好降低噪音数据影响。
        3. 所以本质上是求 $$arg: max_{w, b} \frac{1}{||w||} $$；也就说，我们约束(前提)条件是: $$label*(w^Tx+b)=1$$
- 新的目标函数求解: $$arg: max_{w, b} \frac{1}{||w||} $$
    - => 就是求: $$arg: min_{w, b} ||w|| $$ (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: $$arg: min_{w, b} (\frac{1}{2}*||w||^2)$$ (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 $$label*(w^Tx+b)$$
    - 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: $$L(w,b,\alpha)=\frac{1}{2} * ||w||^2 + \sum_{i=1}^{n} \alpha_i * [1 - label * (w^Tx+b)]$$
    - 因为: $$label*(w^Tx+b)>=1, \alpha>=0$$, 所以 $$\alpha*[1-label*(w^Tx+b)]<=0$$, $$\sum_{i=1}^{n} \alpha_i * [1-label*(w^Tx+b)]<=0$$
    - 当 $$label*(w^Tx+b)>1$$ 则 $$\alpha=0$$ ，表示该点为**非支持向量**
    - 相当于求解: $$max_{\alpha} L(w,b,\alpha) = \frac{1}{2} *||w||^2$$
    - 如果求: $$min*{w, b} \frac{1}{2} *||w||^2$$, 也就是要求: $$min*{w, b} \left(max_{\alpha} L(w,b,\alpha)\right)$$
- 现在转化到对偶问题的求解
    - $$min*{w, b} \left(max*{\alpha} L(w,b,\alpha) \right) $$ >= $$max*{\alpha} \left(min*{w, b}\ L(w,b,\alpha) \right) $$
    - 现在分2步
    - 先求: $$min*{w, b} L(w,b,\alpha)=\frac{1}{2} * ||w||^2 + \sum*{i=1}^{n} \alpha_i * [1 - label * (w^Tx+b)]$$
    - 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>

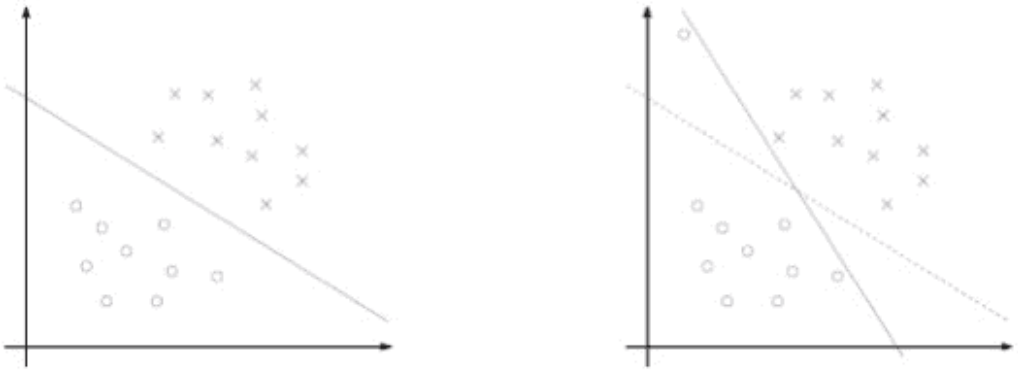


- 终于得到课本上的公式: $$max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i, j=1}^{m} label_i \ast label_j \ast \alpha_i \ast \alpha_j \ast <x_i, x_j> \right) $$
- 约束条件: $$a>=0$$ 并且 $$\sum_{i=1}^{m} a_i \ast label_i=0$$

> 松弛变量(slack variable)

参考地址: http://blog.csdn.net/wusecaiyun/article/details/49659183

看下面两张图:



$$\min_{\gamma,w,b} \ \frac{1}{2}\|w\|^2+C\sum_{i=1}^{m}\xi_i$$
$$s.t. \ \ y^{(i)}(w^T x^{(i)}+b)\geq 1-\xi_i, \ \ i=1,\ldots,m$$
$$\xi_i\geq 0, \ \ i=1,\ldots,m$$

- 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 $$\sum_{i=1}^{m} a_i \ast label_i=0$$
- 总的来说:
  -  表示 `松弛变量`
  - 常量C是 `惩罚因子`, 表示离群点的权重（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” ）
    - $$label*(w^Tx+b) > 1$$ and alpha = 0 (在边界外，就是非支持向量)
    - $$label*(w^Tx+b) = 1$$ and 0< alpha < C (在分割超平面上，就支持向量)
    - $$label*(w^Tx+b) < 1$$ and alpha = C (在分割超平面内，是误差点 -> C表示它该受到的惩罚因子程度)
    - 参考地址: https://www.zhihu.com/question/48351234/answer/110486455
  - C值越大，表示离群点影响越大，就越容易过度拟合；反之有可能欠拟合。
  - 我们看到，目标函数控制了离群点的数目和程度，使大部分样本点仍然遵守限制条件。
  - 例如: 正类有10000个样本，而负类只给了100个（C越大表示100个负样本的影响越大，就会出现过度拟合，所以C决定了负样本对模型拟合程度的影响！，C就是一个非常关键的优化点！）
- 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: $$\sum_{i=1}^{m} a_i \ast label_i=0$$；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。
```

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。

第7章 集成算法

集成算法 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

集成算法 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:

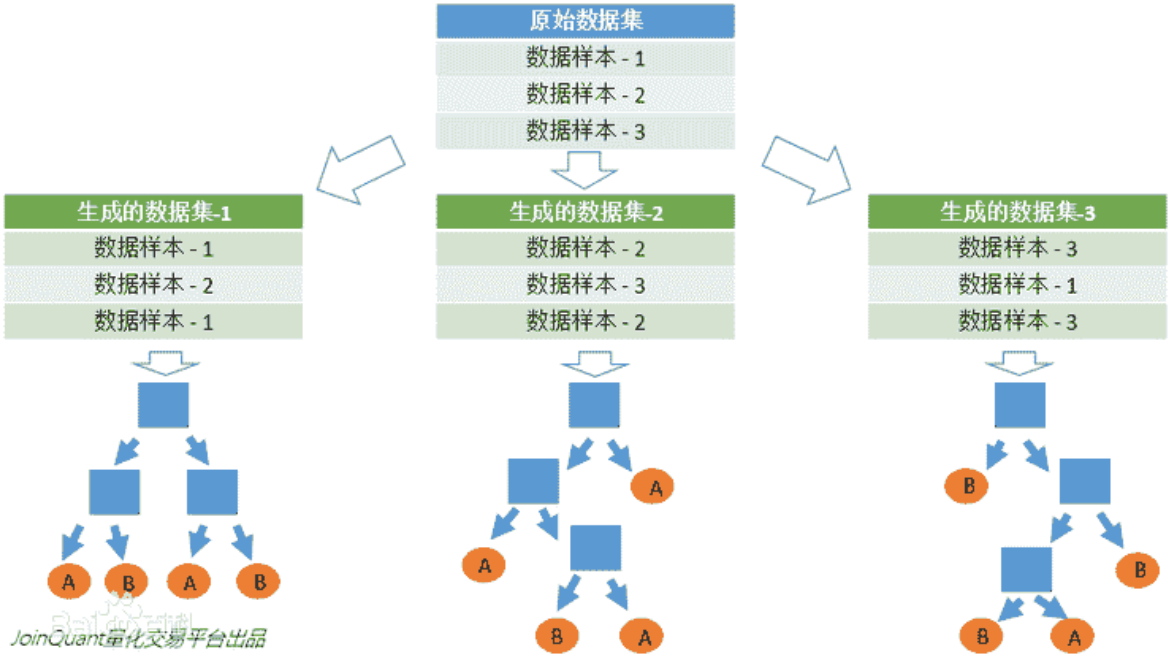
1. 数据的随机性化
2. 待选特征的随机化

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。



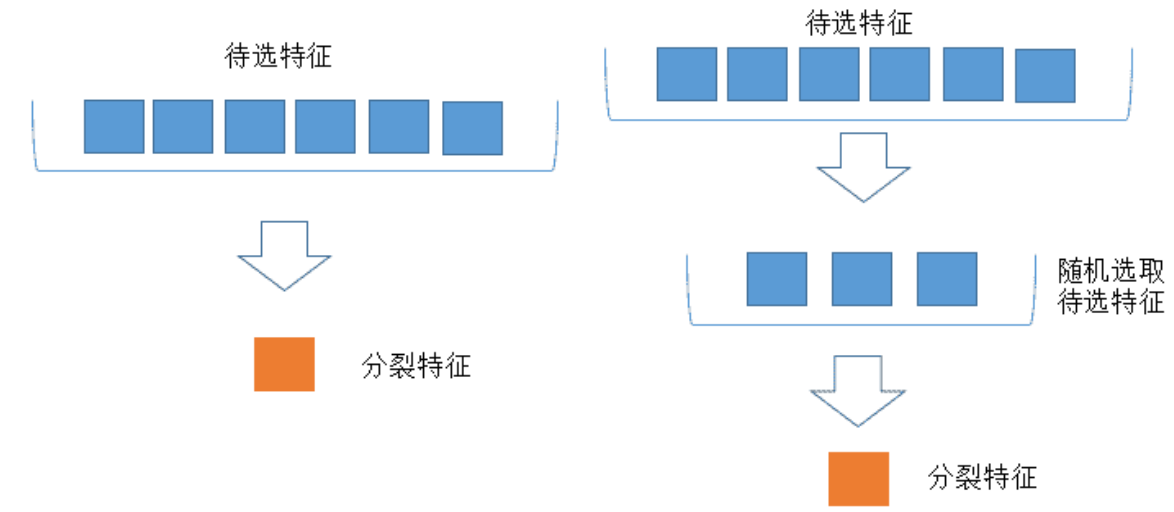
> 待选特征的随机化

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
```

AdaBoost

AdaBoost 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

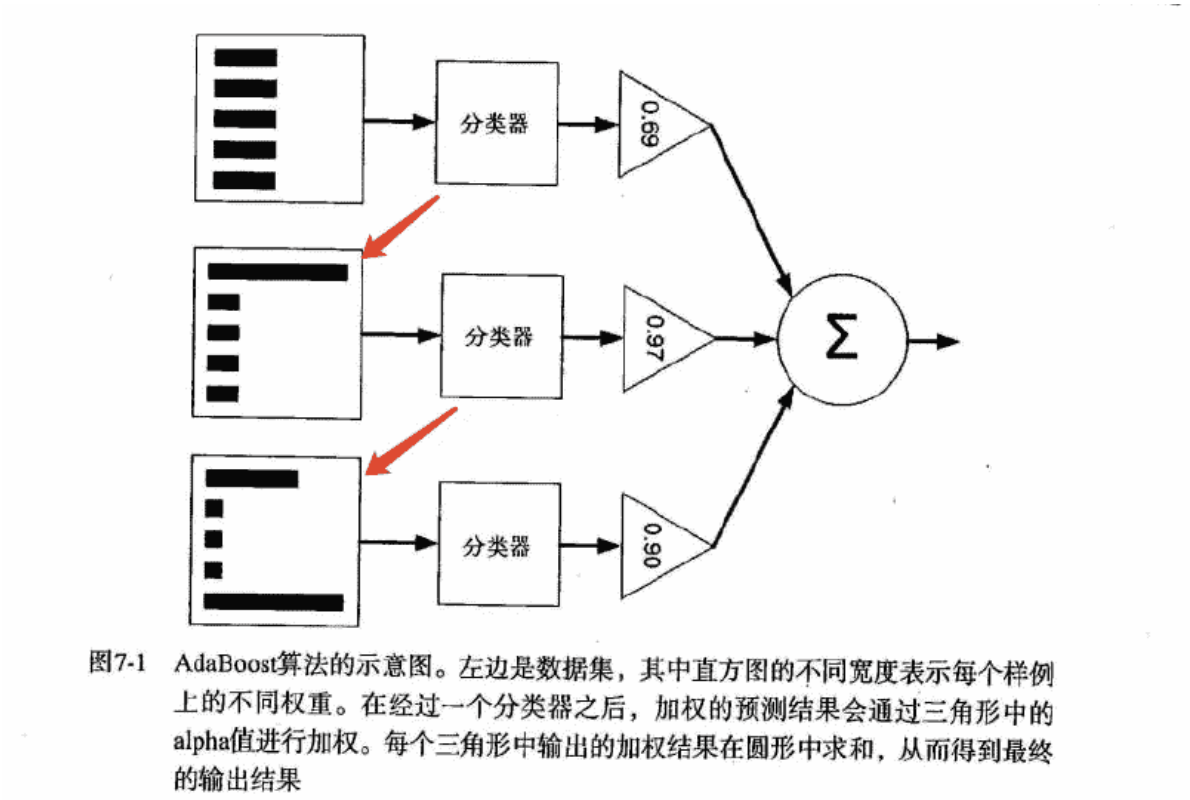


图7-1 AdaBoost算法的示意图。左边是数据集，其中直方图的不同宽度表示每个样例上的不同权重。在经过一个分类器之后，加权的预测结果会通过三角形中的alpha值进行加权。每个三角形中输出的加权结果在圆形中求和，从而得到最终的输出结果

> 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。
```

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）。或者发生在正负例分类错误的成本不同的时候。

- 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

我们有多种方法来处理这个问题: 具体可参考 **此链接**

再结合书中的方法，可以归为八大类:

1.能否收集到更多的数据?

这个措施往往被人们所忽略，被认为很蠢。但是更大的数据集更能体现样本的分布，多样性。

2.尝试使用其他的评价指标

Accuracy 或者error rate 不能用于非均衡的数据集。这会误导人。这时候可以尝试其他的评价指标。

Confusion Matrix 混淆矩阵: 使用一个表格对分类器所预测的类别与其真实的类别的样本统计，分别为: TP、FN、FP与TN。

Precision: 精确度

Recall: 召回率

F1 Score (or F-Score): 精确度和召回率的加权平均

或者使用

Kappa (Cohen's kappa)

ROC Curves

> 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。

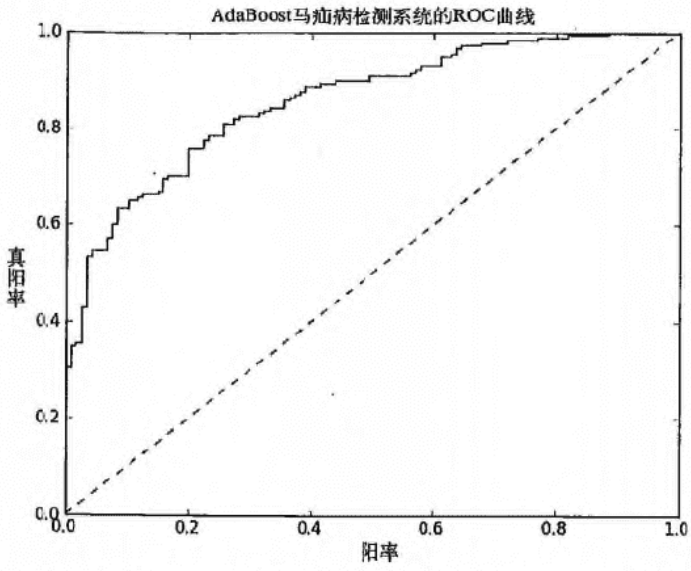


图7-3 利用10个单层决策树的AdaBoost马疝病检测系统的ROC曲线

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。

3.尝试对样本重抽样

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)
```

对大类进行欠抽样

对小类进行过抽样

或者结合上述两种方法进行抽样

一些经验法则:

- 考虑样本（超过1万、十万甚至更多）进行欠采样，即删除部分样本；
- 考虑样本（不足1为甚至更少）进行过采样，即添加部分样本的副本；
- 考虑尝试随机采样与非随机采样两种采样方法；
- 考虑对各类别尝试不同的采样比例，不一定是1:1
- 考虑同时使用过采样与欠采样

4.尝试产生人工生成的样本

一种简单的方法就是随机抽样小类样本的属性（特征）来组成新的样本即属性值随机采样。你可以根据经验进行抽样，可以使用其他方式比如朴素贝叶斯方法假设各属性之间互相独立进行采样，这样便可得到更多的数据，但是无法保证属性之间的非线性关系。

当然也有系统性的算法。最常用的SMOTE(Synthetic Minority Over-Sampling Technique)。 顾名思义，这是一种over sampling（过抽样）的方式。它是产生人为的样本而不是制造样本副本。这个算法是选取2个或者2个以上相似的样本（根据距离度量 distance measure），然后每次选择其中一个样本，并随机选择一定数量的邻居样本对选择的那个样本的一个属性增加噪声(每次只处理一个属性)。这样就构造了更多的新生数据。具体可以参见 **原始论文** 。

python实现可以查阅 UnbalancedDataset

5.尝试不同的算法

强烈建议不要在每个问题上使用你最喜欢的算法。虽然这个算法带来较好的效果，但是它也会蒙蔽你观察数据内蕴含的其他的信息。至少你得在同一个问题上试试各种算法。具体可以参阅 Why you should be Spot-Checking Algorithms on your Machine Learning Problems

比如说，决策树经常在非均衡数据集上表现良好。创建分类树时候使用基于类变量的划分规则强制使类别表达出来。如果有疑惑，可以尝试一些流行的决策树，比如, C4.5, C5.0, CART 和 Random Forrest。

6.尝试使用惩罚的模型

你可以使用同种算法但是以不同的角度对待这个问题。

惩罚的模型就是对于不同的分类错误给予不同的代价（惩罚）。比如对于错分的小类给予更高的代价。这种方式会使模型偏差，更加关注小类。

通常来说这种代价/惩罚或者比重在学习中算法是特定的。比如使用代价函数来实现:

> 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	−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	−1
真实结果	+1	−5	1
	−1	50	0

这种方式叫做 cost sensitive learning，Weka 中相应的框架可以实现叫 CostSensitiveClassifier

如果当你只能使用特定算法而且无法重抽样，或者模型效果不行，这时候使用惩罚（penalization）是可行的方法。这提供另外一种方式来“平衡”类别。但是设定惩罚函数/代价函数是比较复杂的。最好还是尝试不同的代价函数组合来得到最优效果。

7.尝试使用不同的角度

其实有很多研究关于非均衡数据。他们有自己的算法，度量，术语。

从它们的角度看看你的问题，思考你的问题，说不定会有新的想法。

两个领域您可以考虑: anomaly detection(异常值检测) 和 change detection（变化趋势检测）。

Anomaly dectection 就是检测稀有事件。 比如通过机器震动来识别机器谷中或者根据一系列系统的调用来检测恶意操作。与常规操作相比，这些事件是罕见的。

把小类想成异常类这种转变可能会帮助你想到新办法来分类数据样本。

change detection 变化趋势检测类似于异常值检测。但是他不是寻找异常值而是寻找变化或区别。比如通过使用模式或者银行交易记录来观察用户行为转变。

这些两种转变可能会给你新的方式去思考你的问题和新的技术去尝试。

8.尝试去创新

仔细思考你的问题然后想想看如何将这问题细分为几个更切实际的小问题。

比如:

将你的大类分解成多个较小的类;

使用One Class分类器（看待成异常点检测）;

对数据集进行抽样成多个数据集，使用集成方式，训练多个分类器，然后联合这些分类器进行分类;

第8章 回归

回归 概述

> 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。

补充:
线性回归假设特征和结果满足线性关系。其实线性关系的表达能力非常强大，每个特征对结果的影响强弱可以由前面的参数体现，而且每个特征变量可以首先映射到一个函数，然后再参与线性计算。这样就可以表达特征与结果之间的非线性关系。

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（实际上就是我们通常所说的最小二乘法）。

平方误差可以写做（其实我们是使用这个函数作为 loss function）:

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

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

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

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:

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

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。
```

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:

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

目标函数中 w 为权重，不是回归系数。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:

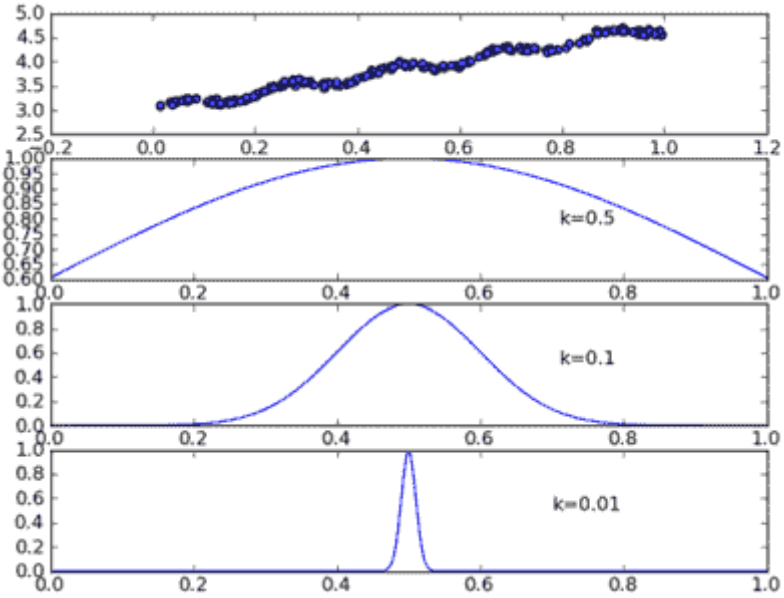
$$\hat{w} = (X^T W X)^{-1} X^T W y$$

其中 W 是一个矩阵，用来给每个数据点赋予权重。$\hat{w}$ 则为回归系数。 这两个是不同的概念，请勿混用。

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:

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

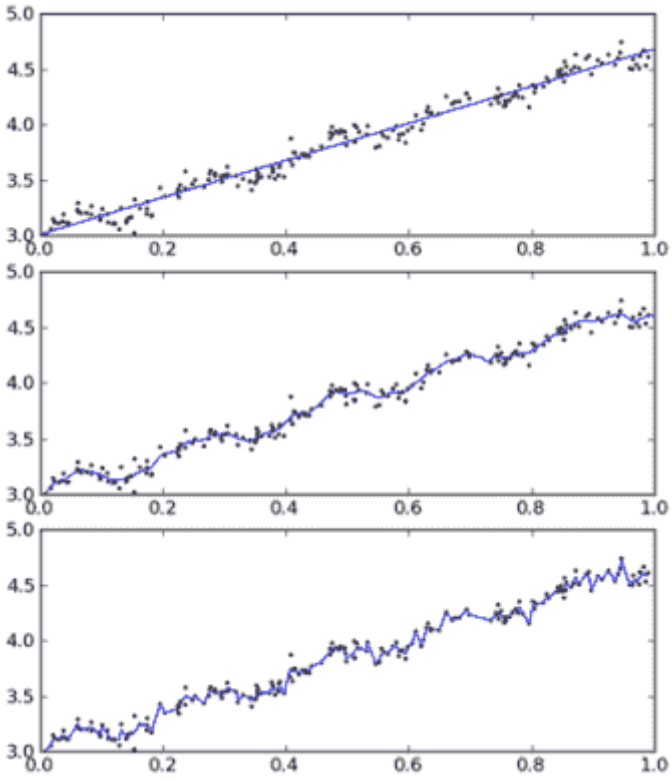
这样就构建了一个只含对角元素的权重矩阵 **w**，并且点 x 与 x(i) 越近，w(i) 将会越大。上述公式中包含一个需要用户指定的参数 k ，它决定了对附近的点赋予多大的权重，这也是使用 LWLR 时唯一需要考虑的参数，下面的图给出了参数 k 与权重的关系。



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 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、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` 。

3.1、岭回归

简单来说，岭回归就是在矩阵 $\mathbf{X}^T\mathbf{X}$ 上加一个 λI 从而使得矩阵非奇异，进而能对 $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ 求逆。其中矩阵I是一个 n * n （等于列数） 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

$$\theta = (X^T X)^{-1} X^T y$$

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，$X^T X$ 的行列式接近于0，即 $X^T X$ 接近于奇异，上述问题变为一个不适定问题，此时，计算 $(X^T X)^{-1}$ 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：

$$\theta(\alpha) = (X^T X + \alpha I)^{-1} X^T y$$

（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 。

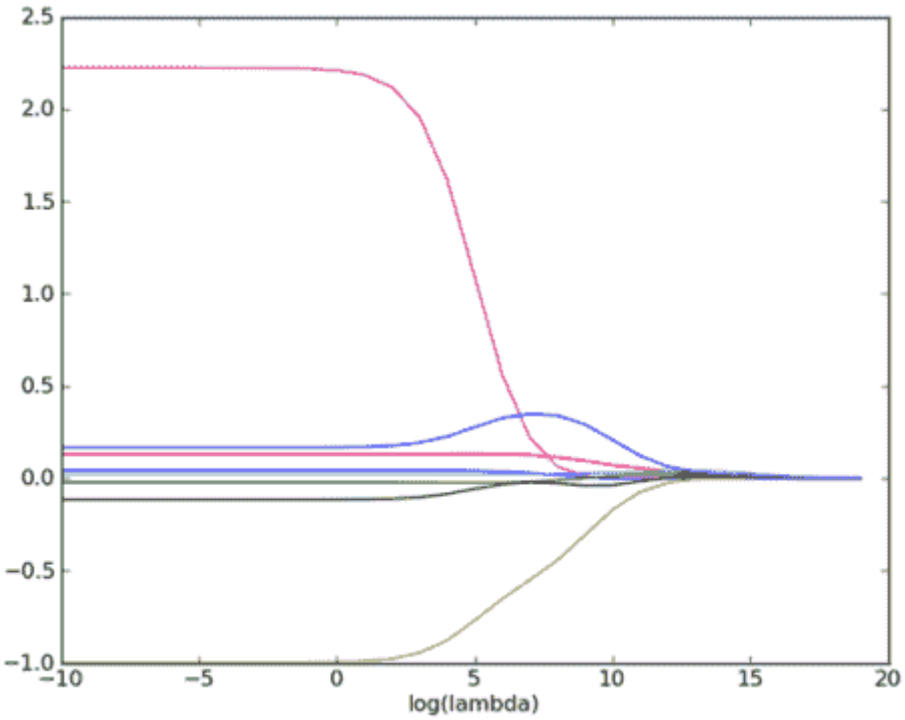
4.1.1、岭回归 原始代码

完整代码地址：https://github.com/apachecn/AiLearning/blob/master/src/py2.x/ml/8.Regression/regression.py

```
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线性回归了，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("data/8.Regression/abalone.txt")
        ridgeWeights = ridgeTest(abX, abY)
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(ridgeWeights)
        plt.show()
```

4.1.2、岭回归在鲍鱼数据集上的运行效果



上图绘制出了回归系数与 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。

3.2、套索方法(Lasso，The Least Absolute Shrinkage and Selection Operator)

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。正是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。

3.3、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 0，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。

3.4、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。

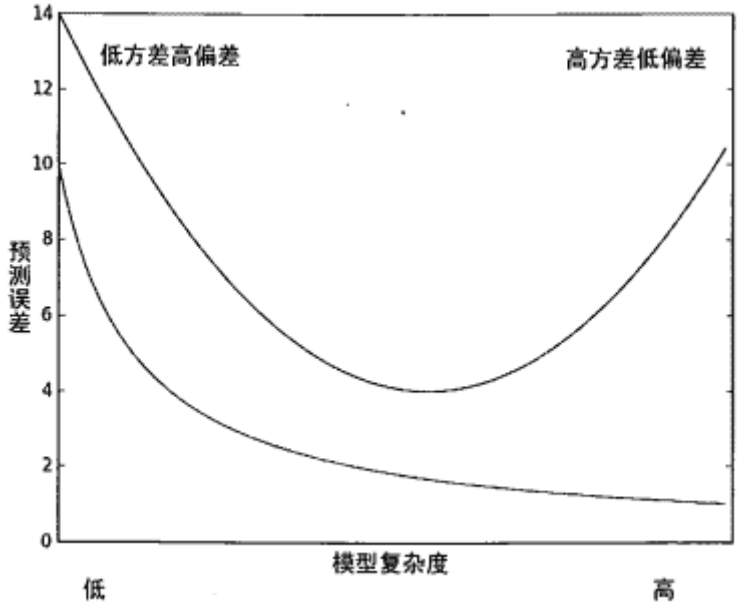
4、权衡偏差和方差

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。我们尝试过仅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。另外一种方法是二元切分法，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，二元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 做分类工作时，采用 GINI 值作为节点分裂的依据；回归时，采用样本的最小方差作为节点的分裂依据。

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

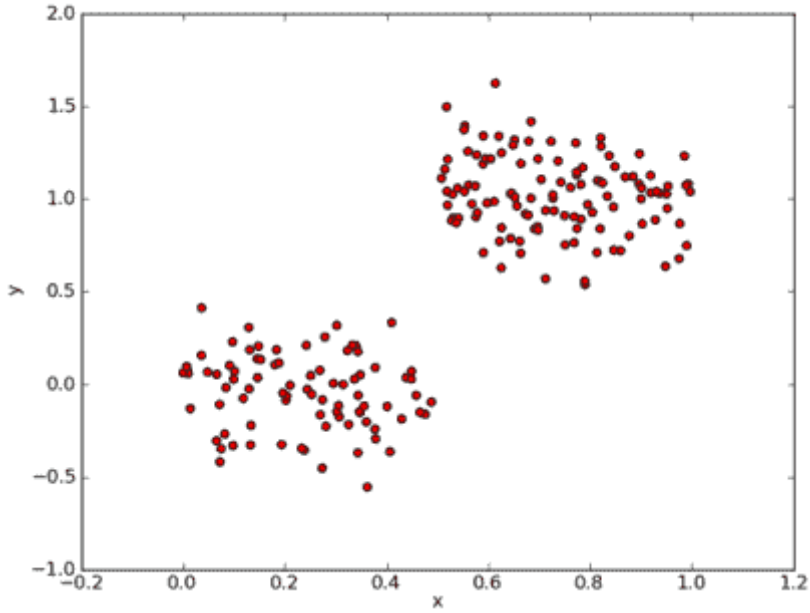
data1.txt 文件中存储的数据格式如下:

```
0.036098    0.155096
0.993349    1.077553
0.530897    0.893462
0.712386    0.564858
0.343554    -0.371700
0.098016    -0.332760
```

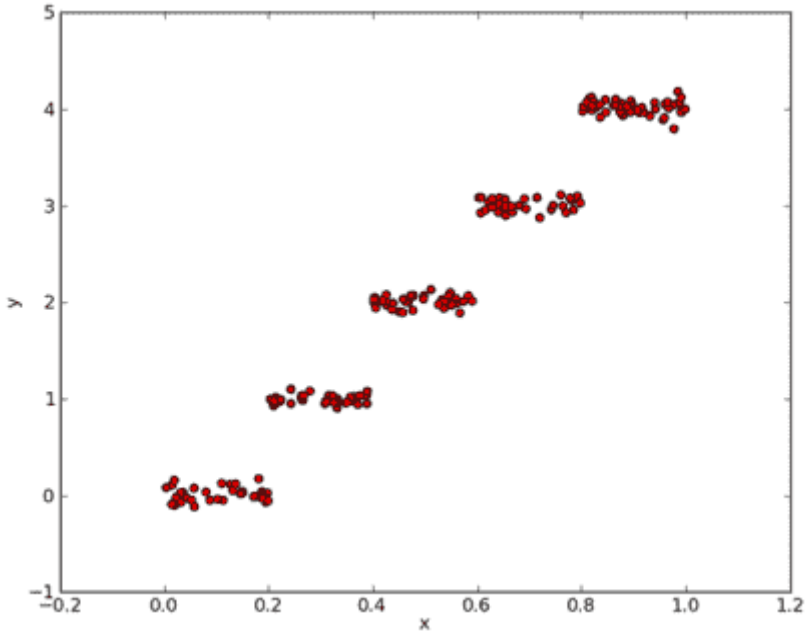
> 准备数据: 需要数值型数据，标称型数据应该映射成二值型数据

> 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/AiLearning/blob/master/src/py2.x/ml/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。

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。

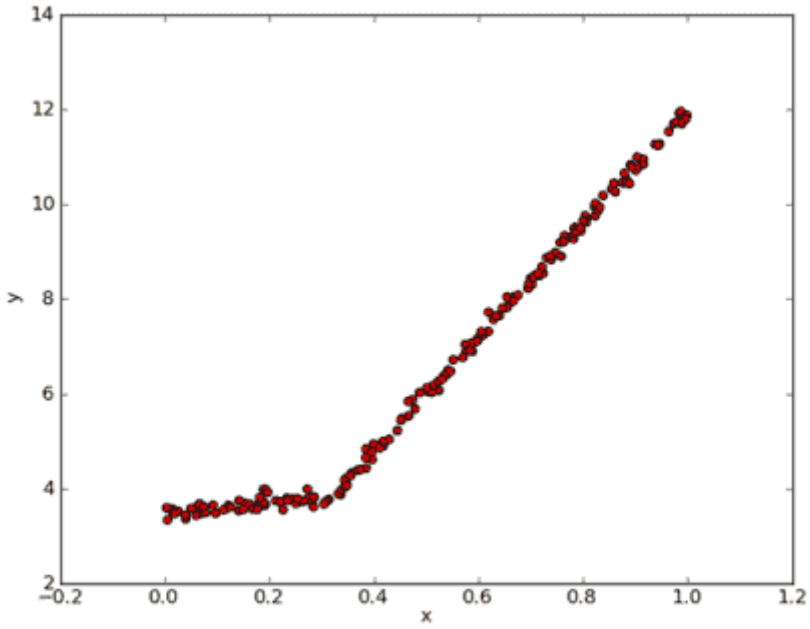


Figure 9.4 **Piecewise linear data to test model tree-creation functions**

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。

第10章 K-Means

聚类

聚类，简单来说，就是将一个庞杂数据集中具有相似特征的数据自动归类到一起，称为一个簇，簇内的对象越相似，聚类的效果越好。它是一种无监督的学习(Unsupervised Learning)方法,不需要预先标注好的训练集。聚类与分类最大的区别就是分类的目标事先已知，例如猫狗识别，你在分类之前已经预先知道要将它分为猫、狗两个种类；而在你聚类之前，你对你的目标是未知的，同样以动物为例，对于一个动物集来说，你并不清楚这个数据集内部有多少种类的动物，你能做的只是利用聚类方法将它自动按照特征分为多类，然后人为给出这个聚类结果的定义（即簇识别）。例如，你将一个动物集分为了三簇（类），然后通过观察这三类动物的特征，你为每一个簇起一个名字，如大象、狗、猫等，这就是聚类的基本思想。

至于“相似”这一概念，是利用距离这个评价标准来衡量的，我们通过计算对象与对象之间的距离远近来判断它们是否属于同一类别，即是否是同一个簇。至于距离如何计算，科学家们提出了许多种距离的计算方法，其中欧式距离是最为简单和常用的，除此之外还有曼哈顿距离和余弦相似性距离等。

欧式距离，我想大家再熟悉不过了，但为免有一些基础薄弱的同学，在此再说明一下，它的定义为:
对于x点坐标为(x1,x2,x3,...,xn)和 y点坐标为(y1,y2,y3,...,yn)，两者的欧式距离为:

$$d(x,y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

在二维平面，它就是我们初中时就学过的两点距离公式

K-Means 算法

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:

- 属于无监督学习，无须准备训练集
- 原理简单，实现起来较为容易
- 结果可解释性较好

缺点:

- **需手动设置k值。** 在算法开始预测之前，我们需要手动设置k值，即估计数据大概的类别个数，不合理的k值会使结果缺乏解释性
- 可能收敛到局部最小值, 在大规模数据集上收敛较慢
- 对于异常点、离群点敏感

使用数据类型 : 数值型数据

K-Means 场景

kmeans，如前所述，用于数据集内种类属性不明晰，希望能够通过数据挖掘出或自动归类出有相似特点的对象的场景。其商业界的应用场景一般为挖掘出具有相似特点的潜在客户群体以便公司能够重点研究、对症下药。

例如，在2000年和2004年的美国总统大选中，候选人的得票数比较接近或者说非常接近。任一候选人得到的普选票数的最大百分比为50.7%而最小百分比为47.9% 如果1%的选民将手中的选票投向另外的候选人，那么选举结果就会截然不同。 实际上，如果妥善加以引导与吸引，少部分选民就会转换立场。尽管这类选举者占的比例较低，但当候选人的选票接近时，这些人的立场无疑会对选举结果产生非常大的影响。如何找出这类选民，以及如何在有限的预算下采取措施来吸引他们？ 答案就是聚类（Clustering)。

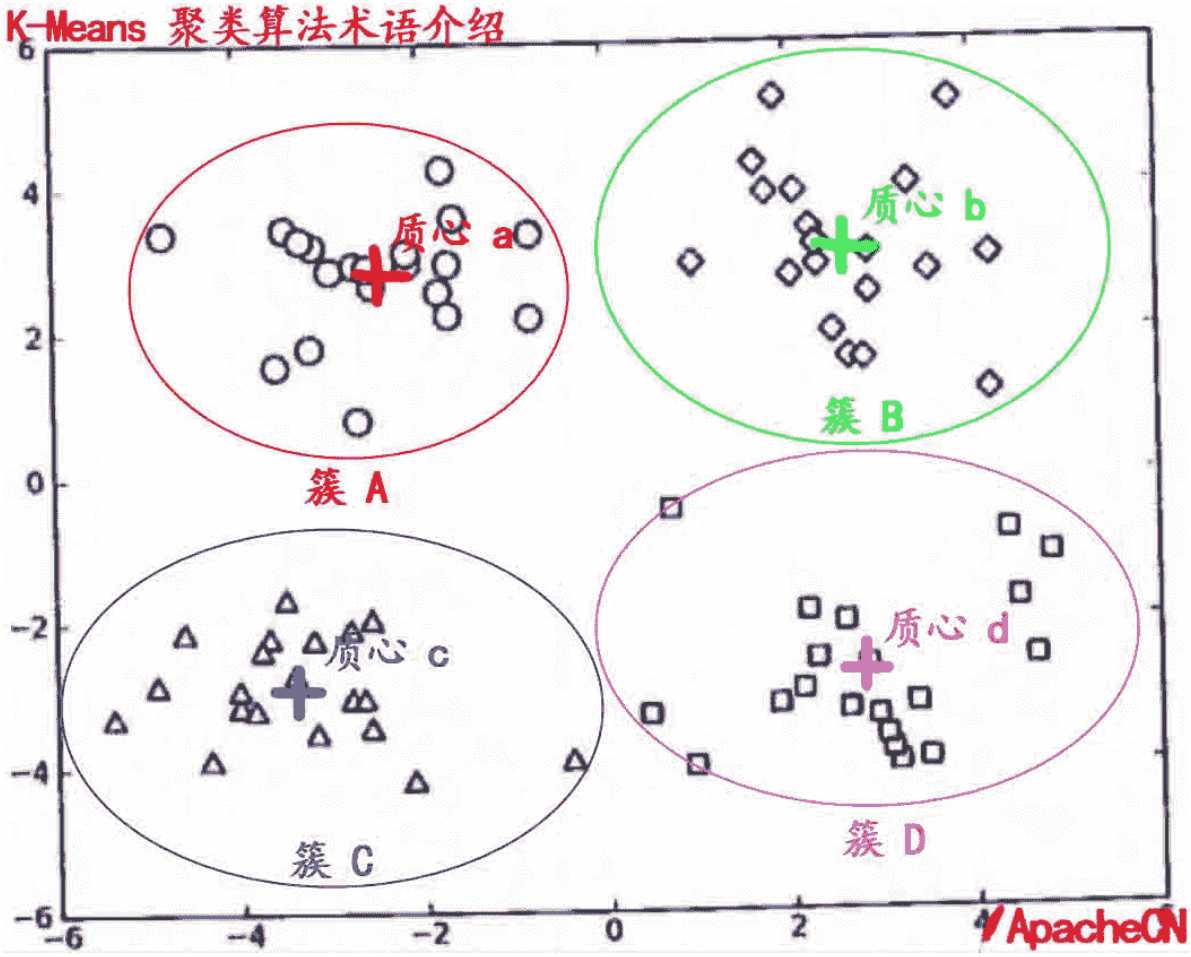
那么，具体如何实施呢？首先，收集用户的信息，可以同时收集用户满意或不满意的信息，这是因为任何对用户重要的内容都可能影响用户的投票结果。然后，将这些信息输入到某个聚类算法中。接着，对聚类结果中的每一个簇（最好选择最大簇 ）， 精心构造能够吸引该簇选民的消息。最后， 开展竞选活动并观察上述做法是否有效。

另一个例子就是产品部门的市场调研了。为了更好的了解自己的用户，产品部门可以采用聚类的方法得到不同特征的用户群体，然后针对不同的用户群体可以对症下药，为他们提供更加精准有效的服务。

K-Means 术语

- 簇: 所有数据的点集合，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（误差平方和）, 它被用来评估模型的好坏，SSE 值越小，表示越接近它们的质心. 聚类效果越好。由于对误差取了平方，因此更加注重那些远离中心的点（一般为边界点或离群点）。详情见kmeans的评价标准。

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（**不必是数据中的点**）。
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所有点的平均值.
   3.重复上述过程直到数据集中的所有点都距离它所对应的质心最近时结束。

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（不改变时算法结束）
  - 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：不适用于无监督学习，即无监督学习不需要训练步骤
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 的评价标准

k-means算法因为手动选取k值和初始化随机质心的缘故，每一次的结果不会完全一样，而且由于手动选取k值，我们需要知道我们选取的k值是否合理，聚类效果好不好，那么如何来评价某一次的聚类效果呢？也许将它们画在图上直接观察是最好的办法，但现实是，我们的数据不会仅仅只有两个特征，一般来说都有十几个特征，而观察十几维的空间对我们来说是一个无法完成的任务。因此，我们需要一个公式来帮助我们判断聚类的性能，这个公式就是**SSE** (Sum of Squared Error, 误差平方和），它其实就是每一个点到其簇内质心的距离的平方值的总和，这个数值对应kmeans函数中**clusterAssment**矩阵的第一列之和。 **SSE**值越小表示数据点越接近于它们的质心，聚类效果也越好。 因为对误差取了平方，因此更加重视那些远离中心的点。一种肯定可以降低**SSE**值的方法是增加簇的个数，但这违背了聚类的目标。聚类的目标是在保持簇数目不变的情况下提高簇的质量。