# 机器学习实战

## 第一部分 分类

主要讨论：监督学习。

提供：输入样本集。机器推演：指定目标变量可能结果。

目标变量：1)标称型，在有限目标集中取值【真 假】【猫 狗】

2)数值型用于回归分析，在书中第二部分

前七章分类算法

第二章 最简单的分类算法：k-近邻算法，用距离矩阵分类

第三章 决策树：直观容易理解难以实现

第四章 用概率论建立分类器

第五章 Logistic回归 用最优参数正确分类原始数据，搜索最优参数时将使用几个常用的优化算法

第六章 支持向量机

第七章 元算法AdaBoost 由若干个分类器构成 总结第一部分分类算法在实际使用中面对的非均衡分类问题：训练样本某个分类数据多于其他分类的数据，就会产生非均衡分类问题。

### 机器学习基础

分类：1)算法训练，学习如何分类。输入大量已分类数据作为算法训练集。目标变量是机器学习算法的预测结果，分类算法中通常为标称型，回归算法中通常是连续型。将分类问题中的目标变量称为类别。2)测试机器学习算法的效果：独立样本集训练数据和测试数据。知识表示可以采用规则集的形式也可以采用概率分布的形式。

主要任务：

1)分类。将实例数据划分到合适分类中。

2)回归。预测数值型数据。

以上两者是监督学习，知道要预测什么，即目标变量的分类信息。

3)聚类。将数据集合分成有类似对象组成的多个类。

4)密度统计。寻找描述数据统计值。

5)减少数据特征的维度。二维表示三维。

无监督学习，没有类别信息也不会给定目标值。

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；K-均值：最大期望算法；DBSCAN：Parzen窗设计。

选择算法：1.目的，2.分析或收集的数据类型

考虑数据特性：特征值是离散型变量还是连续型变量，特征之中是否存在缺失的值，数据中是否存在异常值，某个数据发生的频率如何等。

开发程序步骤：1)收集数据。2)准备输入数据。3)分析输入数据。4)训练算法。5)测试算法。6)使用算法

Python介绍。

### 第二章 k-近邻算法

#### 2.1 k-近邻算法概述

采用测量不同特征值之间的距离方法分类

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。

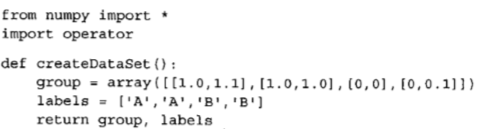
适用数据范围：数值型和标称型。

k-近邻算法(kNN)工作原理：存在一个样本数据集合，也称作训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。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，然后算法提取样本集中特征最相似数据（最近邻）的分类标签。一般只选择前k个最相似数据，k不大于20，并选择出现次数最多的分类作为新数据的分类。

(1)收集数据。(2)准备数据，距离计算所需要的数值。(3)分析数据。(4)训练算法，k-近邻算法不适用。(5)测试算法，算法错误率。(6)使用算法，输入样本和结构化输出结果，运行算法判断数据属于哪个分类。

##### 2.1.1 准备：使用Python导入数据

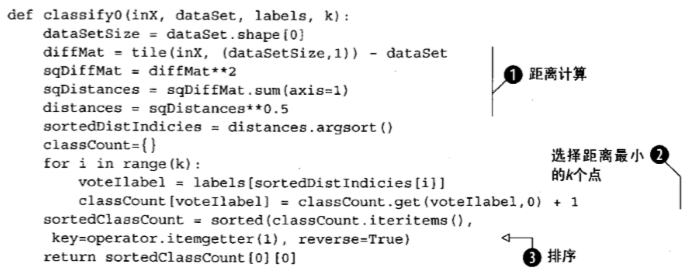
[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch02\kNN.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch02\kNN.py)



##### 2.1.2 从文本文件中解析数据

对未知类别属性的数据集中的每个点依次执行以下操作：

1. 计算已知类别数据集中的点与当前点之间的距离；
2. 按照距离递增次序排序；
3. 选取与之前点距离最小的k个点；
4. 确定前k个点所在类别的出现概率；
5. 返回前k个点出现频率最高的类别作为当前点的预测分类。



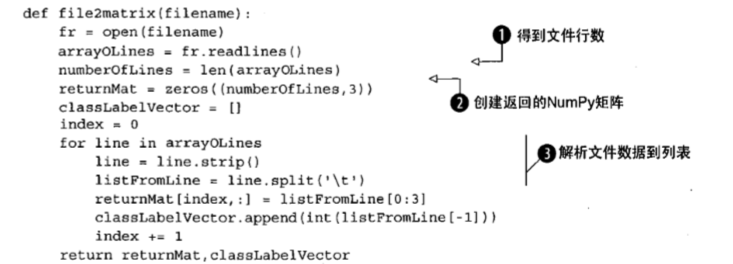
测试输入

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch02\inputa.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch02\inputa.py)

##### 2.1.3 如何测试分类器

错误率 给出错误结果的次数除以测试执行的总数。

整理txt文件信息到矩阵形式



#### 2.2 约会网站配对

##### 2.2.2 使用Matplotlib分析数据

我自己写了输入，修改了Knn模块，输入如下：

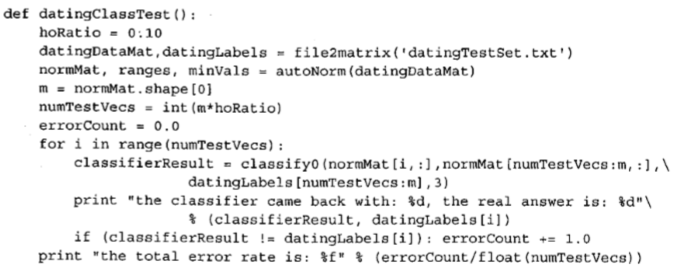
[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch02\inputb.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch02\inputb.py)

##### 2.2.3 准备数据：归一化数据

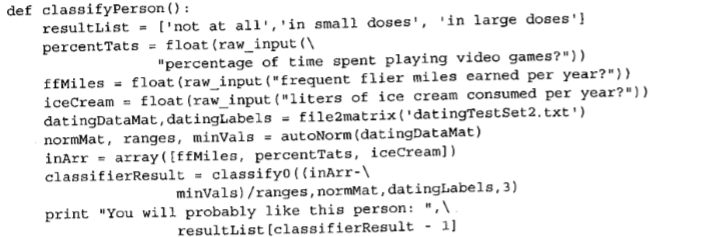
让每种属性的影响相同，需要把所有值的取值范围放在0~1之间或-1~1之间。

##### 2.2.4 测试算法：作为完整程序验证分类器

一般用90%训练。代码在datingClassTest



##### 2.2.5 使用算法：构建完整可用系统

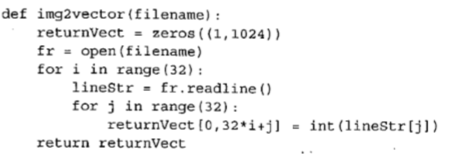


调用函数就一句话

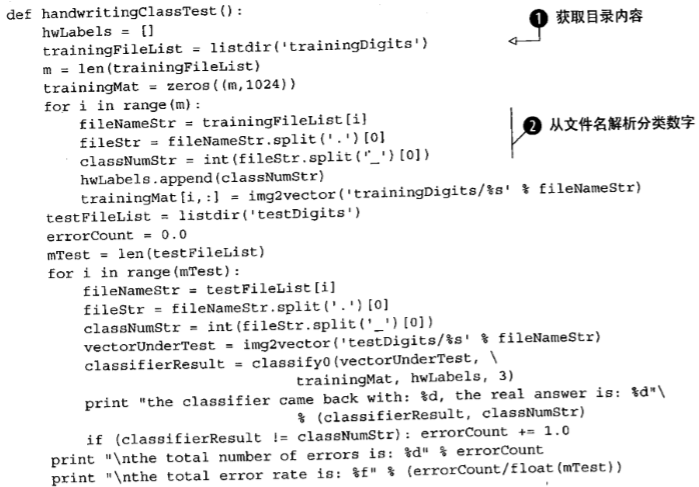
[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch02\inputc.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch02\inputc.py)

#### 2.3 手写识别系统

先将32\*32的图像转化成1\*1024的向量



测试算法



自己写了使用算法函数，调用：

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch02\inpute.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch02\inpute.py)

实际使用算法效率低。

为每个测试向量做n此距离计算

每个距离计算包含m维浮点运算

测试需要存储空间等于运算结果数量

为了减少储存空间和计算时间的开销：k决策树

## 第三章 决策树

k-近邻算法无法给出数据内在含义，决策树主要优势在于数据形式非常容易理解。

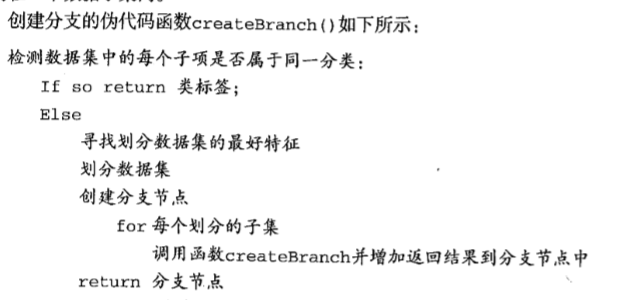
专家系统常使用决策树。

### 3.1 决策树的构造

决策树优点：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。

创建分支：



一般流程：

1. 收集数据：可以使用任何方法；
2. 准备数据：树构造算法只适用于标称型数据，数值型数据必须离散化。
3. 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。
4. 训练算法：构造树的数据结构
5. 测试算法：使用经验树计算错误率
6. 使用算法：适用于任何监督学习算法，使用决策树可以更好理解数据内在含义。

ID3算法划分数据集，进一步信息参考<http://en.wikipedia.org/wiki/ID3_algorithm>。每次划分选择一个特征属性。

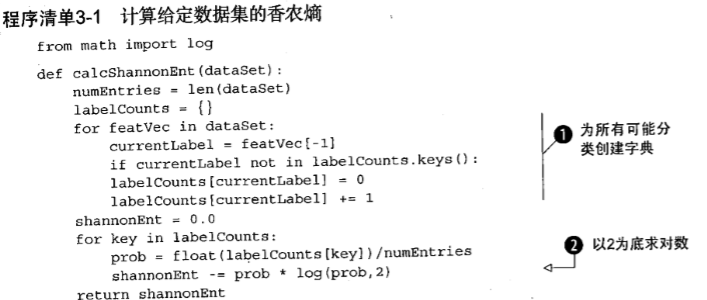
##### 3.1.1 信息增益

集合信息的度量方式称为香农熵简称为熵。熵定义为信息的期望值。待分类的事物划分在多个分类之中，则符号的信息定义为

其中是选择该分类的概率。

为了计算熵，要计算所有类别所有可能值包含的信息期望值：

计算代码：



[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\trees.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\trees.py)

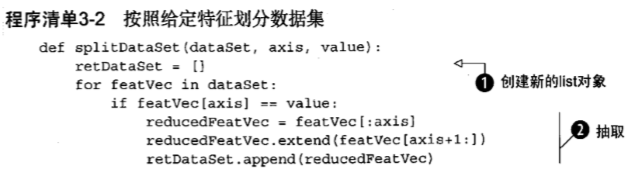
熵越高则混合数据越多。整个测试和加入数据测试在inputa.py中

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputa.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputa.py)

另一个度量集合无序程度的方法是基尼不纯度，简单地说是从一个数据集中随机选取子项，度量其被错误分类到其他分组里的概率。

##### 3.1.2 划分数据集

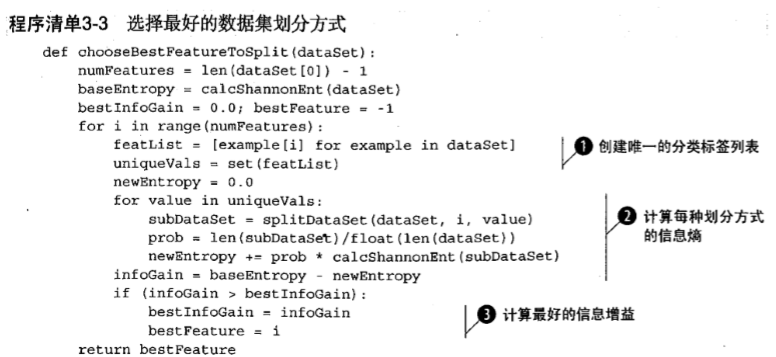
分类算法除了需要测量信息熵，还需要划分数据集，度量花费数据集的熵，判断是否正确划分了数据集。



测试上面的数据集划分：

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputb.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputb.py)

遍历整个数据集，循环计算香农熵和数据计划分找到最好的划分方式。



测试：

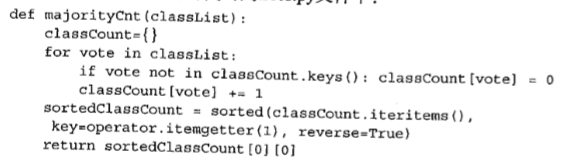
[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputc.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputc.py)

##### 3.1.3 递归构建决策树

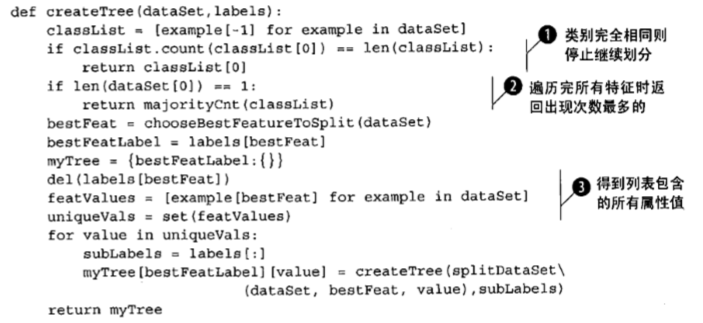
从原始数据集出发，找最好属性值，划分数据集，第一次划分后数据将被向下传递到树分支的下一个节点，在这个节点上可以再次划分数据。可以采用递归的原则处理数据集。

结束条件：遍历完所有划分数据集的属性或每个分支下所有实例具有相同的分类。任何到达叶子节点的数据必然属于叶子节点的分类。

后面还有其他决策树算法如C4.5或CART。在运行时并不总是在每次划分分组时消耗特征。由于特征数目并不是在每次划分数组分组时都减少，这些算法实际应用可能有一些问题。如果数据集已经处理了所有属性，但是类标签不是唯一的，用多数表决方式决定分类。



以上为投票算法。下面给出树的生成算法：



先写结束条件。然后写正常条件：寻找，拆数据集，减特征，最后进循环。

测试创建树：

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputd.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputd.py)

#### 3.2 用Matplotlib注解绘制树形图

注解工具annotations在数据图形上添加文本注解，画带箭头的文字描述

代码如下绘制树节点：

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\treePlotter.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\treePlotter.py)

这个测试被迷之改过了，所以先不做示例输出

然后获得叶节点数目和树的高度。准备了两棵生成好的树到retrieveTree。然后添加计算坐标和绘制的代码。加载测试：

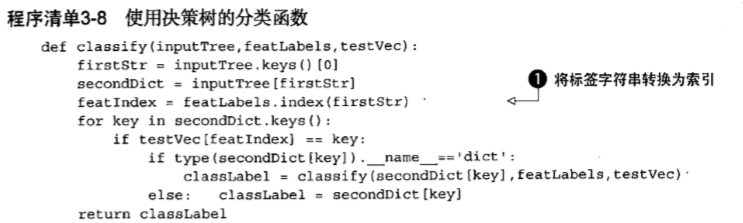
[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inpute.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inpute.py)

#### 3.3 测试和存储分类器

##### 3.3.1 测试算法：使用决策树执行分类

获得决策树后可以用于实际数据的分类。执行数据分类时需要决策树以及用于构造树的标签向量。然后程序比较测试数据与决策树上的数值递归执行该过程进入叶子节点，最后定义测试数据为叶子节点所属类型。

回到trees.py

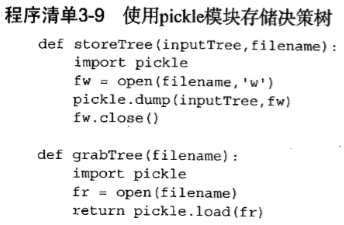


测试代码如下：

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputf.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputf.py)

##### 3.3.2 使用算法：决策树的存储

树的构造代价大，所以先存好用的时候取就好了



事实上我都改成二进制之后才能正常写入读出。测试代码如下：

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputg.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputg.py)

其实应该写数据结构来存树的，不过这样一弄快速性也够了

#### 3.4 示例：使用决策树预测隐形眼镜类型

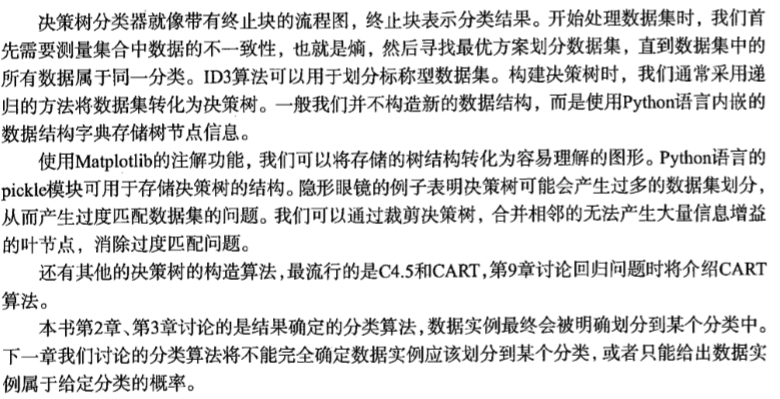
1. 收集数据：提供的文本文件；
2. 准备数据：解析tab分割的数据行。
3. 分析数据： 用createPlot()绘制树形图检查数据内容正确。
4. 训练算法：使用createTree()
5. 测试算法：用测试函数验证其可以正确分类给定的数据实例
6. 使用算法：存储树的数据结构之后不用再造

我还编写了一行测试代码。

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch03\inputh.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch03\inputh.py)

书里说匹配选项太多过渡匹配，裁剪决策树在第九章，处理掉增加少量信息的叶子节点。以及学习另一个决策树构造算法CARTT。ID3无法直接处理数值型数据，以及如果存在太多特征划分会面临其他问题。

#### 3.5 小结



下一章讨论不能完全确定数据示例应该划分到某个分类或智能给出属于特定分类概率的算法。

### 第四章 基于概率论的分类方法：朴素贝叶斯

本章给出一些使用概率论进行分类的方法。从一个简单的概率分类器开始给出一些假设来学习朴素贝叶斯分类器。称之为“朴素”是因为整个形式化过程只做最原始、最简单的假设。

#### 4.1基于贝叶斯决策理论的分类方法

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比较敏感

适用数据类型：标称型数据。

两类数据，通过样本得到测试数据属于每种类型的概率，选择较大的那种。

#### 4.2 条件概率

描述一件事在特定情况下发生的概率：

#### 4.3 使用条件概率分类

分类准则：

如果p1(x,y)>p2(x,y)，输入点属于类别1;

如果p1(x,y)<p2(x,y)，输入点属于类别2。

真正需要计算和比较的是和，即给定坐标点，则其来自类别的概率是多少。进一步可以得到：

判别准则为：

如果，则属于类别1；

如果，则属于类别2。

#### 4.4 使用朴素贝叶斯进行文档分类

朴素贝叶斯一般过程：

1. 收集数据：可用任何方法，本章使用RSS源
2. 准备数据：需要数值型或布尔型数据
3. 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
4. 训练算法：计算不同的独立特征的条件概率
5. 测试算法：计算错误率
6. 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。

加入每个特征需要N个样本，那么m个特征就需要个样本，数量迅速增长。

若特征之间相互独立可以减少到个样本，独立是指一个特征出现的可能性和其他特征没有关系。例如单词A出现在B或C后面的概率相同，事实上单词之间互有联系，这个假设并不正确，但这个假设正是朴素贝叶斯分类器中朴素一次的含义。朴素贝叶斯分类器的另一个假设是每个特征同等重要，这个假设也有问题。一个文档类型看10~20个特征就可以了。尽管存在以上问题，朴素贝叶斯的实际效果却很好。

#### 4.5 使用Python进行文本分类

拆分文本：特征来自文本的词条（token），一个词是字符的任意组合。每一个文本片段标识为一个词条向量，其中值为1表示词条出现在文档中，0表示未出现。

先将文本转换为数字向量，然后介绍如何基于这些向量计算条件概率，并在此基础上构建分类器，最后包括利用Python实现朴素贝叶斯过程中需要考虑的问题。

##### 4.5.1 准备数据：从文本中构建词向量

文件存储在bayes.py

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch04\bayes.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch04\bayes.py)

包括创建列表，提取单词（英文的毕竟太简单了还不考虑词组）转换成集合，以及分析输入文本返回向量。

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch04\inputa.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch04\inputa.py)

上面是测试用的代码。

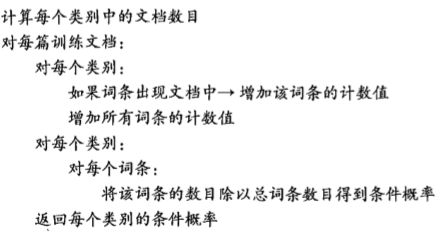
##### 4.5.2 训练算法：从词向量计算概率

这里词向量是多元数组，因此用代替，公式转化为：

1)用有侮辱性词汇的文档数除以总的文档数计算概率;

2)计算，利用朴素贝叶斯假设，将展开为一个个独立特征，那么就可以将上述概率写作。假设所有词相互独立，意味着可以用来计算。

伪代码：



代码也很简单，而且利用python特性实现列表直接相加。测试代码文件：

文档里被改成各种其他参数，估计后面有其他任务。

[Machine Learning in Action\Code for Machine Learning in Action\machinelearninginaction\Ch04\inputb.py](file:///C:\Users\dominatorX\AppData\Roaming\Microsoft\Word\Machine%20Learning%20in%20Action\Code%20for%20Machine%20Learning%20in%20Action\machinelearninginaction\Ch04\inputb.py)

##### 4.5.3 测试算法：根据现实情况修改分类器

没想到这么快就遇到这个修改了……为了防止由于一个概率值为0导致最后乘积为0，给所有词出现数初始化为1，分母初始化为2。为了防止下溢出，取对数保证精度。

因为

对于任意都有相等，因此比较大小等价于比较大小，同样相当于比较

文件里的代码间接实现了这种表达，但这种转换是否合理我还是保持质疑。

##### 4.5.4 准备数据：文档词袋模型

词集模型：将每个词出现与否作为一个特征

词袋模型：每个单词可以出现一次或多次或者不出现。

#### 4.6 示例：使用朴素贝叶斯过滤垃圾邮件

1)收集数据：提供文本文件

2)准备数据：将文本文件解析成词条向量

3)分析数据：检查词条确保解析的正确性

4)训练算法：使用我们之前建立的trainNB0函数

5)测试算法：使用classifyNB()，并且构建一个新的测试函数来计算文档集的错误率

6)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

##### 4.6.1 准备数据：切分文本

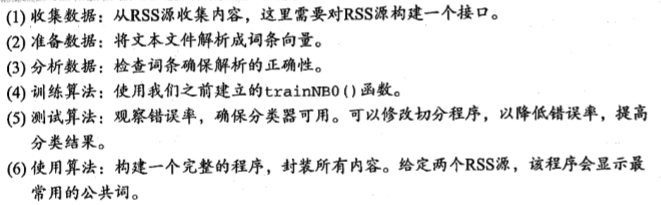
word.split()等等，实际上要去掉文件后缀名，所以过滤掉过短的词以及转换成相同大小写。

##### 4.6.2 测试算法：使用朴素贝叶斯进行交叉验证

写了一个50个训练用例的邮件过滤器，每次随机生成被测编号，一整套训练加测试用之前的模型，只是外加了文档处理功能。

#### 4.7 示例：使用朴素贝叶斯分类器从个人广告中获取区域倾向

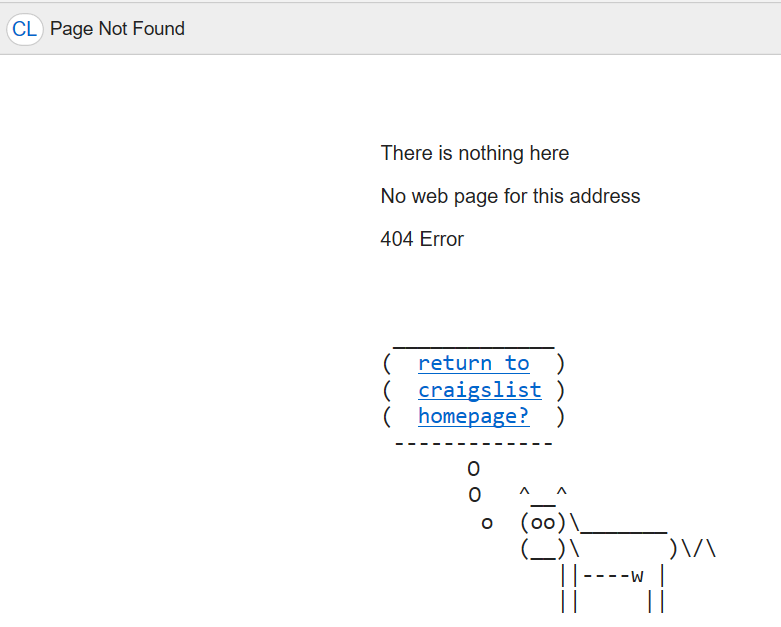
使用朴素贝叶斯发现地域相关的用词



##### 4.7.1 收集数据：导入RSS源

用Python下载文本，利用RSS很容易得到文本。RSS阅读器需要Universal Feed Parser 这个RSS程序库

这里我并不能提取到那两个网站的信息，直接登录网站显示的是：



还挺萌的。

#### 4.8 小结

条件独立性假设

朴素

下溢

词袋

移除停用词

# NLP courses

## C1 Language Modeling

### 1.1 introduction

Constructing a language model

Corpus: a set of sentences in some language

Definition:

: the set of all the words in the language

A sentence in the language is a sequence of words:

n is such that , we have for , and we assume that is a special symbol, STOP. Example sentences could be:

We will deﬁne to be the set of all sentences with the vocabulary : this is an inﬁnite set, because sentences can be of any length.

Definition 1 (Language Model) A language model consists of a finite set , and a function such that:

1.For any ,

2.In addition,

Hence is a probability distribution over the sentences in

For this example, a really simple way to define the function for a corpus with N sentences is use the number of times that is seen in the corpus by:

and none of the sentences that are not in the corpus will have . The key technical contribution of this chapter will be to introduce methods that do generalize to sentences that are not seen in our training data.

### 1.2 Markov Models

#### 1.2.1 Markov Models for Fixed-length Sequences

This is to define the length of the sentence is some fixed number (e.g., n=100). This would be generalized to a random n in next section.

We would like to model the probability of any sequence , that is, to model the joint probability:

There are possible sequences of the form .

In a first-order Markov process, we make the following assumption, which considerably simplifies the model:

(1.1)

(1.2)

In Eq.1.1 the equation is always make sense.

In Eq.1.2 is not necessarily exact.

This is a first-order Markov process. We have assumed that the identity of the ith word in the sequence depends only on the identity of the previous word.

In a second-order Markov process, which will form the basis of trigram language models, we make a slightly weaker assumption, namely that each word depends on the previous two words in the sequence:

It follows that the probability of an entire sequence is written as:

(1.3)

#### 1.2.2 Markov Sequences for Variable-length Sentences

One solution is to add a special symbol STOP at the end of the sentence. And we can use exactly the same assumptions as before: such as a second-order Markov assumption:

(1.4)

A little more formally, the process that generates sentences would be as follows:

1. Initialize , and

2. Generate from the distribution

3. If then return the sequence . Otherwise, set and return to step 2.

Thus we have a model that generates sequences that vary in length.

### 1.3 Trigram Language Models

In this section we give the basic definition of a trigram model, discuss maximum-likelihood parameter estimates for trigram models, and finally discuss strengths of weaknesses of trigram models.

#### 1.3.1 Basic Definitions

Under a second-order Markov model, the probability of any sentence is then

where we assume as before that .

We will assume that for any , for any ,

The model can take the form:

For any bigram u, v:

The key problem is to estimate the parameters of the model, namely:

There are around parameters in the model.

1.3.2 Maximum=Likelihood Estimates

This is the most generic solution to the estimation problem.

First, some notations. Define to be the number of times that the trigram is seen in the training corpus. And define to be the number of times that the bigram is seen in the corpus. Then:

Two problems:

1) Many of the above estimates will be , due to the count in the numerator being 0.

2) In cases where the denominator is equal to zero.

#### 1.3.3 Evaluating Language Models: Perplexity

To measure the quality of a language model, a very common method is to evaluate the perplexity of the model on some held-out data.

Assume that we have some test data sentences .

Each test sentence is a sequence of words .

Every sentence ends up with STOP symbol.

Test sentences are “held out”, new, unseen sentences.

For any test sentence , we can measure the probability . A natural measure of the quality of the language model would be the probability it assigns to the entire set of test sentences, that is

The intuition is as follows: the higher this quantity is, the better the language model is at modeling unseen sentences.

The perplexity on the test corpus is derived as a direct transformation of this quantity. Define M to be the total number of words in the test corpus. More precisely, under the definition that is the length of the test sentence:

Then the average log probability under the model is defined as:

The higher this quantity is, the better the language model.

The perplexity is defined as:

Where:

The smaller the value of perplexity, the better the language model is at modeling unseen data.

Some intuitions:

A vocabulary , where ，and the model predicts

for all u, v, w. Thus this is the dumb model that simply predicts the uniform distribution over the vocabulary together with the STOP symbol. So we can see that the perplexity is equal to N. So under a uniform probability model, the perplexity is equal to the vocabulary size. Perplexity can be thought of as the effective vocabulary size under the model.

The perplexity is equal to:

where:

Damnnnnnn it!

is just the geometric mean of the terms

If any trigram u, v, w seen in test data, we have the estimate:

then the perplexity will be ∞. Thus if we want to take perplexity seriously as our measure of a language model, then we should avoid giving 0 estimates at all costs.

For a trigram model the perplexity figures is much more less than bigram and unigram.

#### 1.3.4 Strengths and Weaknesses of Trigram Language Models

Quite strong and linguistically naïve

Very useful in practice.

### 1.4 Smoothed Estimation of Trigram Models

The maximum-likelihood parameter estimates:

Will run into serious issues with sparse data.

We discuss two smoothing methods that very commonly used in practice: first, linear interpolation; second, discounting methods.

#### 1.4.1 Linear Interpolation

We define the trigram, bigram, and unigram maximum-likelihood estimates as：

where is the total number of words seen in the training corpus.

The idea in linear interpolation is to use all three estimates, by defining the trigram estimate as follows:

where

and:

There are various ways of estimating the values. A common one is as follows.

We have development data separated with training and testing data. Define to be the number of times that the trigram is seen in the development data. We can see that the log-likelihood of the development can be defined data as:

We would like to choose the which made the as high as possible.

A much simpler method is to define:

where is the only parameter of the method. The can again be chosen by maximizing log-likelihood of a set of development data.

This method is relatively crude, and is not likely to be optimal. It is, however, very simple, and in practice it can work well in some applications.

#### 1.4.2 Discounting Methods

Define discounted count as:

is a value between 0 and 1, typical value of it is . This reflect the intuition that if we take counts from the training corpus, we will systematically over-estimate the probability of bigrams seen in the corpus and under-estimate bigrams not seen in the corpus.

For any bigram , we can define:

And define missing probability mass:

More specifically, the complete definition of the estimate is as follows. For any , define the sets:

and

Then the estimate is defined as:

And obviously for the trigram we have:

where:

Define to be the number of times that the trigram is seen in this development corpus. The log-likely hood of the development data is:

We usually choose the from a set such as , We then choose the value for that maximizes this log-likelihood.

### 1.5 Advanced Topics

#### 1.5.1 Linear Interpolation with Bucketing

In linearly interpolated models the parameter estimates are defined as:

Generally speaking, we should have different smoothing parameters with different number of . A function maps bigrams to values . This function is defined by hand, such as:

We have different parameters for all

Then we have a more precise model:

## C2 Tagging Problems, and Hidden Markov Models

### 2.1 Introduction

To model pairs of sequences, part-of-speech(POS)[词性] tagging is perhaps the earliest, and most famous example of this type of problem.

In POS tagging our goal is to build a model whose input is a sentence, for example:

The dog saw a cat

And whose output is a tag sequence, for example:

D N V D N

We use to denote the input to the tagging model: we will often refer to this as a sentence. The length of it is . We will use to denote the output of the tagging model: we often refer to this as the state sequence or tag sequence.

The problem with the task to map a sentence to a tag sequence is often refereed to as a sequence labeling problem, or a tagging problem.

We have a set of training examples, for . Each sentence equals to , and each is a tag sequence .

### 2.2 Two Example Tagging Problems: POS Tagging, and Named-Entity Recognition

Challenges in POS:

1. Ambiguity 2. The presence of words that are rare

First, individual words have statistical preferences for their part of speech. Second, the context has an important effect on the part of speech for a word.

Entity recognition has an input as a sentence but with a output as entity-boundaries marked, like PERSON, LOCATION, and COMPANY. Each word in the sentence is either tagged as not being part of an entity (the tag NA) is used for this purpose, as being the start of a particular entity type (e.g., the tag SC) corresponds to words that are the first word in a company, or as being the continuation of a particular entity type such as the tag CC corresponds to words that are part of a company name, but are not the first word.

### 2.3 Generative Models, and The Noisy Channel Model

Treat tagging problems as a supervised learning problem.

Generative models are in one important class of supervised learning and a particular type of it named hidden Markov models applied to tagging problem.

The set-up in supervised learning problems is as follows. Training examples like . We use to refer to the set of possible inputs, and refer to the set of possible labels. We need to learn a function .

One way to define the function is through a conditional model. In this approach we define a model that defines the conditional probability:

the output from the model is:

An alternative approach, which is often used in machine learning and natural language processing, is to define a **generative model**. Rather than directly estimating the conditional distribution , in generative models we instead model the joint probability:

which:

is a prior probability distribution over labels

is the probability of generating the input , given that the underlying label is .

According to Bayes rule we have:

where:

for an input , the function can be derived as:

(2.3)

(2.4)

Model that decompose a joint probability into terms and are often called noisy-channel models.

### 2.4 Generative Tagging Models

A finite vocabulary and possible tags . And we give following definition:

**Definition 1 (Generative Tagging Models)** Assume a finite set of words , and a finite set of tags. Define to be the set of all sequence/tag-sequence pairs , such that , for , and for . A generative tagging model is then a function such that:

1. for any ,

2. In addition,

Hence is a probability distribution over pairs of sequences.

Given a generative tag model, the function can be defined as:

### 2.5 Trigram Hidden Markov Models (Trigram HMMs)

#### 2.5.1 Definition of Trigram HMMs

**Definition 2 (Trigram Hidden Markov Model (Trigram HMM)** Assume a finite set of words , and a finite set of tags. Together with following parameters:

For any trigram such that , and , the value for can be interpreted as the probability of seeing the tag immediately after bigram of tags .

For any , . The value for can be interpreted as the probability of seeing observation paired with state .

Define to be the set of all sequence /tag-sequence pairs , where .

We then define the probability for any as:

And we assume that

#### 2.5.2 Independence Assumptions in Trigram HMMs

To derive the above equation, we need to get an assumption that:

First, we can write:

The step is exact. As the last chapter, we have a second-order Markov sequence form for the origin sequence:

And more assumption for the conditional probability:

The steps are as follow:

1.Initialize and .

2.Generate from the distribution:

3.If then return . Otherwise, generate from the distribution:

Set, and return to step 2.

#### 2.5.3 Estimating the Parameters of a Trigram HMM

Define to be the number of times the sequence of three states is seen in training data, as well as the for tag bigram is seen. Define as the number of times that is seen in the corpus. Finally, define to be the number of times state is seen paired with observation .

Then we have the maximum-likelihood estimates as:

and:

And with the **chapter.1** we are smooth the estimates of with:

One more question for the case that or at a low level. A solution is described in section 2.7.1.

#### 2.5.4 Decoding with HMMs: the Viterbi Algorithm

The naïve, brute force method would be to simply enumerate all possible tag sequences and score them under the function:

And take the highest scoring sequence.

This method results in hopelessly inefficient with possible tag sequences.

**The Basic Algorithm**

A dynamic programming algorithm that is often called **the Viterbi algorithm**. We can define the function:

This is simply a truncated version of the definition of is **Eq.2.8**, where we just consider the first terms. In particular, note that:

It will be convenient to use for to denote the set of allowable tags at position in the sequence: more precisely, define:

and

Define to be the set of all tag sequences of length , ending in the tag bigram . Define: