核心算法

那么既然是朴素贝叶斯分类算法，它的核心算法又是什么呢？

是下面这个贝叶斯公式：

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

换个表达形式就会明朗很多，如下：

$$p(\text{类别}|\text{特征}) = \frac{p(\text{特征}|\text{类别})p(\text{类别})}{p(\text{特征})}$$

我们最终求的p(类别|特征)即可！就相当于完成了我们的任务。

例题分析

帅？	性格好？	身高？	上进？	嫁与否
帅	不好	矮	不上进	不嫁
不帅	好	矮	上进	不嫁
帅	好	矮	上进	嫁
不帅	好	高	上进	嫁
帅	不好	矮	上进	不嫁
不帅	不好	矮	不上进	不嫁
帅	好	高	不上进	嫁
不帅	好	高	上进	嫁
帅	好	高	上进	嫁
不帅	不好	高	上进	嫁
帅	好	矮	不上进	不嫁
帅	好	矮	不上进	不嫁

核心算法

我们的问题是，如果一对男女朋友，男生向女生求婚，男生的四个特点分别是不帅，性格不好，身高矮，不上进，请你判断一下女生是嫁还是不嫁？

这是一个典型的分类问题，转为数学问题就是比较p(嫁|(不帅、性格不好、身高矮、不上进))与p(不嫁|(不帅、性格不好、身高矮、不上进))的概率，谁的概率大，我就能给出嫁或者不嫁的答案！

核心算法

这里我们联系到朴素贝叶斯公式：

$$p(\text{嫁}|\text{不帅、性格不好、身高矮、不上进})=\frac{p(\text{不帅、性格不好、身高矮、不上进}|\text{嫁})*p(\text{嫁})}{p(\text{不帅、性格不好、身高矮、不上进})}$$

那么我只要求得p(不帅、性格不好、身高矮、不上进|嫁)、p（不帅、性格不好、身高矮、不上进)、p(嫁)即可

核心算法

p(不帅、性格不好、身高矮、不上进|嫁) = p(不帅|嫁)*p(性格不好|嫁)*p(身高矮|嫁)*p(不上进|嫁)，那么我就要分别统计后面几个概率，也就得到了左边的概率！

为什么这个成立呢？学过概率论的同学可能有感觉了，这个等式成立的条件需要特征之间相互独立吧！

这也就是为什么朴素贝叶斯分类有朴素一词的来源，朴素贝叶斯算法是假设各个特征之间相互独立，那么这个等式就成立了！

例题分析

我们将上面公式整理一下如下：

$$p(\text{嫁}|\text{不帅、性格不好、身高矮、不上进})=\frac{p(\text{不帅、性格不好、身高矮、不上进}|\text{嫁})*p(\text{嫁})}{p(\text{不帅、性格不好、身高矮、不上进})}$$

$$=\frac{p(\text{不帅}|\text{嫁})*p(\text{性格不好}|\text{嫁})*p(\text{身高矮}|\text{嫁})*p(\text{不上进}|\text{嫁})*p(\text{嫁})}{p(\text{不帅})*p(\text{性格不好})*p(\text{身高矮})*p(\text{不上进})}$$

嫁的概率

p(嫁)= ？
首先我们整理训练数据中，嫁的样本数如下

帅？	性格好？	身高？	上进？	嫁与否
帅	好	矮	上进	嫁
不帅	好	高	上进	嫁
帅	好	高	不上进	嫁
不帅	好	中	上进	嫁
帅	好	中	上进	嫁
不帅	不好	高	上进	嫁

则 p(嫁) = 6/12（总样本数） = 1/2

不帅嫁的概率

p(不帅|嫁)=？统计满足样本数如下：

帅？	性格好？	身高？	上进？	嫁与否
不帅	好	高	上进	嫁
不帅	好	中	上进	嫁
不帅	不好	高	上进	嫁

则p(不帅|嫁) = 3/6 = 1/2 在嫁的条件下，看不帅有多少

性格不好嫁的概率

p(性格不好|嫁)=？统计满足样本数如下：

帅？	性格好？	身高？	上进？	嫁与否
不帅	不好	高	上进	嫁

则p(性格不好|嫁)= 1/6

矮和不上进分别嫁的概率

p(矮|嫁)=？统计满足样本数如下：

帅	好	矮	上进	嫁

则p(矮|嫁) = 1/6

p(不上进|嫁) = ?统计满足样本数如下:

帅？	性格好？	身高？	上进？	嫁与否
帅	好	高	不上进	嫁

则p(不上进|嫁) = 1/6

P（不帅），P(性格不好)等

p(性格不好|嫁)=？统计满足样本数如下：

帅？	性格好？	身高？	上进？	嫁与否
帅	不好	矮	不上进	不嫁
不帅	好	矮	上进	不嫁
帅	好	矮	上进	嫁
不帅	好	高	上进	嫁
帅	不好	矮	上进	不嫁
帅	不好	矮	上进	不嫁
帅	好	高	不上进	嫁
不帅	好	中	上进	嫁
帅	好	中	上进	嫁
不帅	不好	高	上进	嫁
帅	好	矮	不上进	不嫁
帅	好	矮	不上进	不嫁

性格不好统计如上红色所示，占4个，那么p（性格不好）= 4/12 = 1/3

身高矮统计如上红色所示，占7个，那么p（身高矮）= 7/12

p（不上进）= 4/12 = 1/3

不帅统计如上红色所示，占4个，那么p（不帅）= 4/12 = 1/3

女生嫁的概率

$$p(\text{嫁}\mid\text{不帅、性格不好、身高矮、不上进})=\frac{p(\text{不帅、性格不好、身高矮、不上进}\mid\text{嫁})*p(\text{嫁})}{p(\text{不帅、性格不好、身高矮、不上进})}$$

$$=\frac{p(\text{不帅}\mid\text{嫁})*p(\text{性格不好}\mid\text{嫁})*p(\text{身高矮}\mid\text{嫁})*p(\text{不上进}\mid\text{嫁})*p(\text{嫁})}{p(\text{不帅})*p(\text{性格不好})*p(\text{身高矮})*p(\text{不上进})}$$

= (1/2*1/6*1/6*1/6*1/2)/(1/3*1/3*7/12*1/3)

女生不嫁的概率

p (不嫁|不帅、性格不好、身高矮、不上进)
= ((1/6*1/2*1*1/2)*1/2)/(1/3*1/3*7/12*1/3)

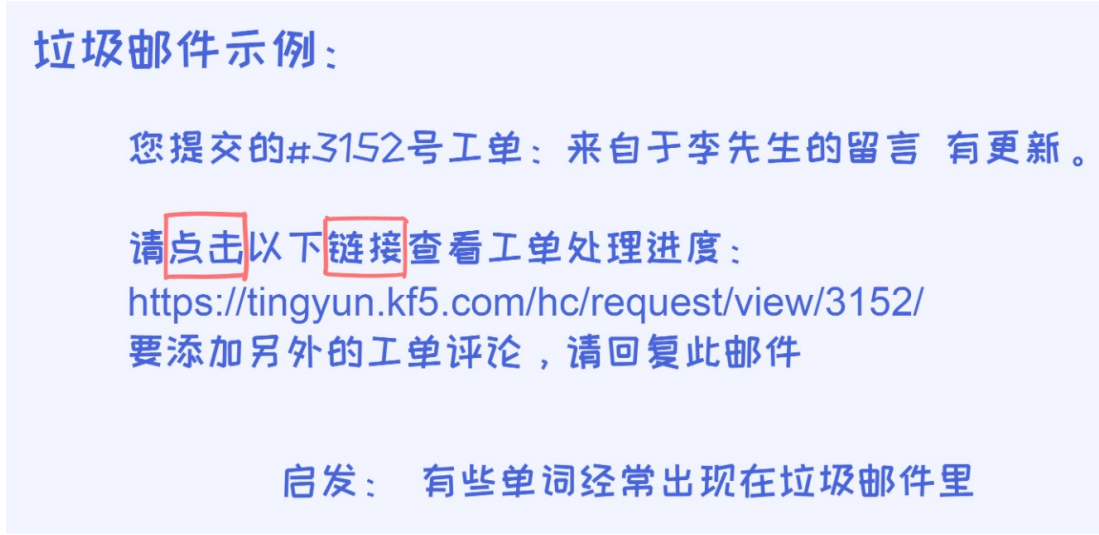
很显然(1/3*1/2*1*2/3) > (1/2*1/6*1/6*1/6)

于是有p (不嫁|不帅、性格不好、身高矮、不上进)>p (嫁|不帅、性格不好、身高矮、不上进)

所以我们根据朴素贝叶斯算法可以给这个女生答案，是不嫁！！！！

垃圾邮件分类

只要遇到了文本分类问题， 第一个需要想到的方法就是朴素贝叶斯，在文本分类上它是一个非常靠谱的基准(baseline)。



算法的核心思想极其简单：其实就是统计出不同文本类别中出现的词频。对于垃圾邮件的分类任务，我们需要统计哪些单词经常出现在垃圾邮件，哪些单词经常出现在正常的邮件里。

如果在邮件里看到了“广告”，“购买”，“链接”等关键词，可以认为这个很可能是个垃圾邮件。 其实很多邮件过滤系统就是这么过滤掉垃圾邮件的。

核心思想

朴素贝叶斯-核心思想

统计单词在不同类别中出现的概率，然后根据这些结果

进一步判断一个文本它属于不同类别的概率

先验概率和后验概率

假设迟到的原因有2个：

1.天气不好

2.起床晚了

先验概率：迟到的概率

条件概率：已知天气不好的条件下，迟到的概率

后验概率：已经迟到了，因为天气原因迟到的概率

垃圾邮件分类

利用朴素贝叶斯识别垃圾邮件

前面计算出了很多零零散散的概率，怎么整合这些信息来完成识别任务？

从概率统计的角度

P(垃圾|邮件内容)：一个邮件内容为垃圾邮件的概率

P(正常|邮件内容)：一个邮件内容为正常邮件的概率

如何做判断？

如果 P(垃圾|邮件内容) > P(正常|邮件内容)，则可以认为是垃圾邮件

如果 P(垃圾|邮件内容) <= P(正常|邮件内容)，则可以认为是正常邮件

垃圾邮件分类

另外，在上述过程中可以看到分子的计算过程涉及到了很多概率的乘积。一旦遇到这种情形，就要知道可能会有潜在的风险。比如其中一个概率等于0， 那不管其他概率值是多少，最后的结果一定会等于0。 那这种情况应该如何处理呢?

平滑处理。接着我给你们说一下比较常用的平滑处理方法，叫作: add-one smoothing。

垃圾邮件分类

$$P(w \mid y=c) = \frac{\text{类别为c的语料库中的单词w出现的次数}+1}{\text{类别为c的语料库中包含的所有单词个数}+v}$$

v为词典的大小

$$P(apple \mid y=1) + P(app \mid y=1) + \cdots + P(zoo \mid y=1) = 1$$

$$V = \begin{bmatrix} apple \\ app \\ \vdots \\ zoo \end{bmatrix}$$

垃圾邮件分类

一个完整的例子

垃圾邮件

1. 点击 更多 信息
2. 最新 产品
3. 信息 点击 链接

正常邮件

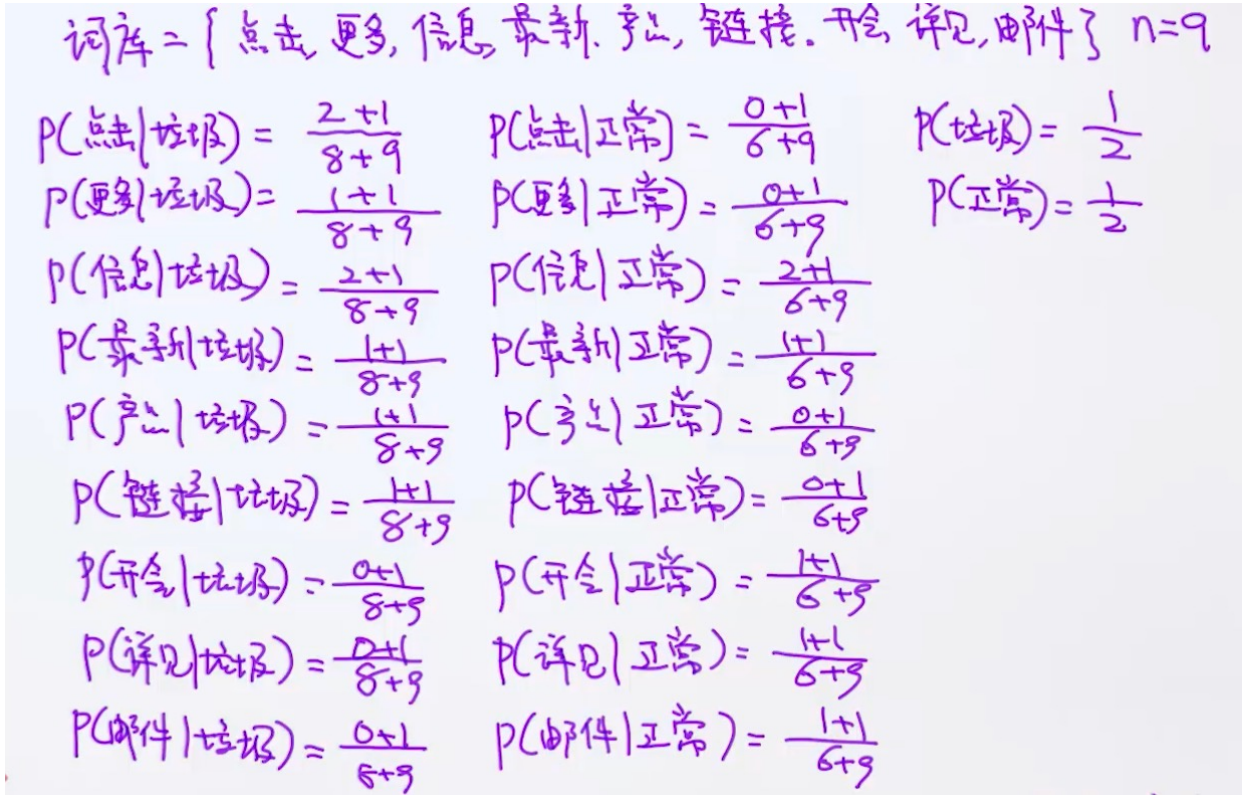
1. 开会
2. 信息 详见 邮件
3. 最新 信息

新邮件

最新 产品 点击 链接 产品

属于正常邮件还是垃圾邮件？

垃圾邮件分类

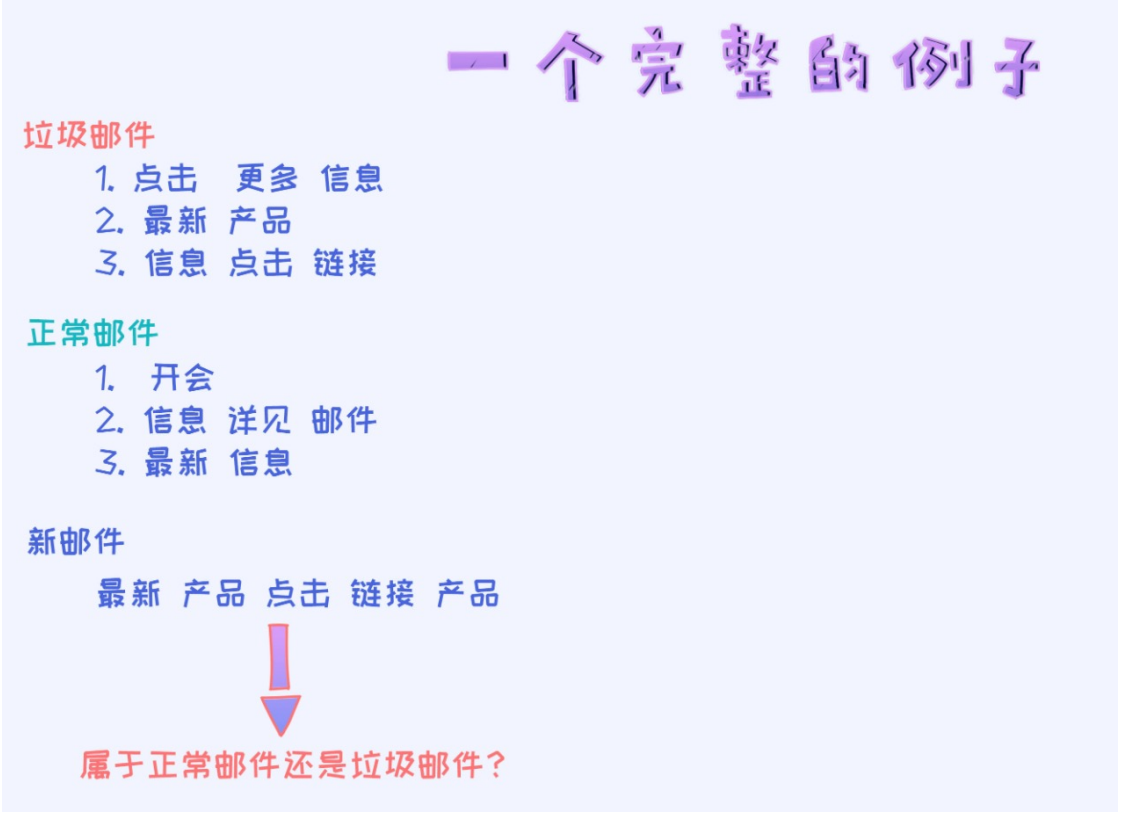


垃圾邮件的概率：2/17 * 2/17 * 3/17 * 2/17 * 2/17= 0.00003381

vs

正常邮件的概率：2/15 * 1/15 * 1/15 * 1/15 * 1/15 = 0.00000263；最终比较的结果是垃圾邮件

最后分子连续相乘容易产生underflow，所以我们一般前面加一个log。
LogP(垃圾) + logP(最新|垃圾) + …..



朴素贝叶斯总结

- 朴素贝叶斯的核心思想是统计单词在不同类别中的概率
- 朴素贝叶斯是最简单且最经典的文本分类算法
- 朴素贝叶斯使用了条件独立的性质，这也是为什么它叫“朴素”的主要原因
- 朴素贝叶斯使用了贝叶斯定理，这也是为什么叫它“贝叶斯”的主要原因
- 为了避免概率值等于0，在朴素贝叶斯训练中会使用平滑技术

Jieba分词

最常用的还是Jieba(结巴)分词，又快又有效。
结巴分词的使用法非常简单。

!pip install jieba

jieba

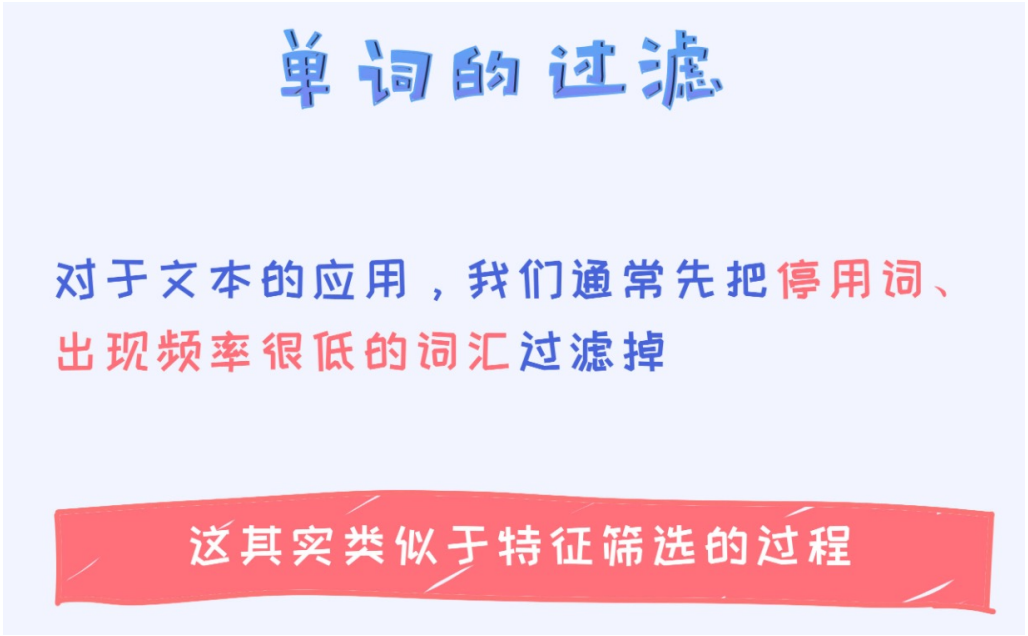
- **1. Jieba 的 3 种分词模式**

Jieba 提供了以下 3 种分词模式。

- 精确模式：试图将句子精确地切开，适合文本分析；
- 全模式：把句子中所有可以成词的词语都扫描出来。全模式处理速度非常快，但是不能解决歧义；
- 搜索引擎模式：在精确模式的基础上，对长词再次切分，提高召回率，适用于搜索引擎分词。

停用词

那什么叫停用词呢？ 其实很容易理解：**就是那些出现特别频繁，但对于一个句子贡献不是特别大的单词**。比如”的“， ”他“可以认为是停用词。



问题

假如我们解决文本分类问题，那文本中的数字如何处理比较合适呢？

1）直接去掉
2）保留并把每一个数看作是一个单词
3）把所有的数字表示为特殊词比如 “#NUM#”
4）以上都不太合适

问题解析

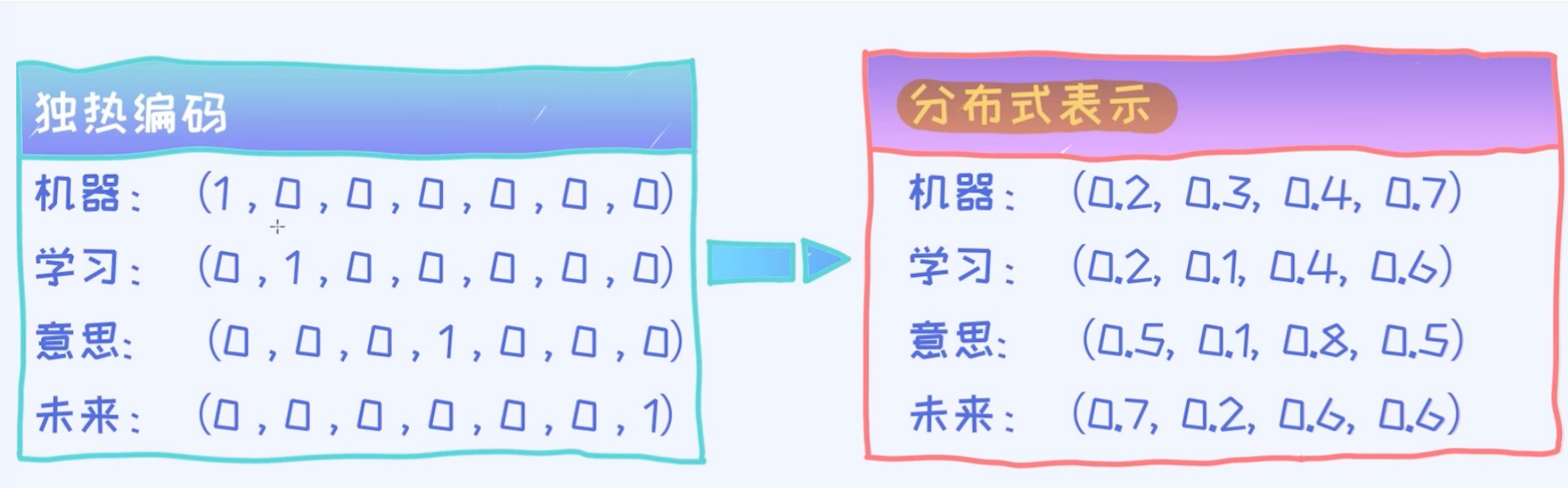
首先，数字本身是有意义的，至少说明这个是一个“数字“，但具体是什么数字其实很难把它表示出来。为什么呢？因为数字不像单词有一个完整的可以提前定义好的库。数字本身是无穷多的，我们没有办法把所有的都列出来。而且具体是什么数对于理解文本来说意义没有那么大。基于这些理由，我们通常把出现的所有的数表示成一个统一的特殊符号比如"#NUM"，这样至少我们的模型知道这是一个数字。

Jieba功能介绍

参看tfidf.ipynb文档

文本向量化

如何把一个文本表示成向量的形式。文本本身属于非结构化数据，而且我们要知道非结构化数据是不能直接作为模型的输入的。



tf-idf

- TF-IDF代表的是词频-逆文档频率，是两个度量的组合。在数学上TFIDF是两个度量的乘积，可以表示为$tfidf = tf * idf$。词频（TF）是一词语出现的次数除以该文件的总词语数。假如一篇文件的总词语数是100个，而词语“程序员”出现了3次，那么“程序员”一词在该文件中的词频就是3/100=0.03。“程序员”的TF值会根据不同的文章求得（每篇文章的TF值可能不同，取决于“程序员”出现的次数以及这篇文章的总词数），以这些TF值乘以后续的IDF值计算这个词在每篇文章中的TFIDF值。

idf

逆向文件频率 (IDF) 指的是每个词语的文档频率的逆。该值由语料库中全部文档数量除以包含每个词语的文档的数目，再将得到的结果取对数得到，值得注意的是，将对每个词语的文档频率加1，意味着词汇表中每个词语至少包含在一个语料库的文档之后，避免分母为0的错误，平滑逆文档频率，最后也对idf的计算结果加1，避免分子与分母相同的时候（总文档数目与某个词语在各篇文章出现次数加1正好相同）得到的对数为0。数学公式如下表示：

$$idf(t) = 1 + log\frac{C}{1 + df(t)}$$

idf

其中idf(t)表示词语t的IDF，C表示语料库中文档的总数量，df(t)表示包含词语t的文档数量频率。如果包含词语t的文档越少，IDF越大，则说明词语具有很好的类别区分能力。如果以上述“程序员”为例子，计算文件频率（IDF）的方法是以文件集的文件总数除以出现“程序员”一词的文件数。假设“程序员”一词在1,000份文件出现过，而文件总数是10,000,000份的话，其逆向文件频率就是log（10,000,000 / 1,000）=4。最后的tf-idf的分数为0.03 * 4=0.12。

Tfidf归一化

另外补充一下，TFIDF的度量还需要做一次归一化，一般使用的是L2范数来进行矩阵归一化（矩阵是tf和idf的乘积，将TFIDF矩阵除以矩阵的L2范数来进行矩阵的归一化，L2范数也称为欧几里得范数，它是每个词语TFIDF的度量（基于每个词本身的所在的文章）求平方和（将每个词语的tfidf值求平方然后相加）然后对这个结果求平方根）。

Sklearn中的tfidf实现

```
# 建立 tf-idf 词频权重矩阵
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(norm='l2') #使用 l2 范数做归一化
tf_train_data = tfidf.fit_transform(train_content) #得到 tfidf 值
```

Sklearn.feature_extreaction.text.tfidfvecotrizer参数

use_idf：布尔值，默认为 True。使用逆文档频率重新加权；smooth_idf：布尔值，默认为 True。通过对文档频率加 1 来平滑 idf 权值。ngram_range 参数类型为 tuple，使用场景为如果觉得单个的词语作为特征还不足够，能够加入一些词组更好，就可以设置这个参数，如下面允许词表使用 1 个词语，或者 2 个词语的组合，使用 ngram_range=(1,2)来表示；例如['一切'，'一切 星球']，得到的 TFIDF 值为['星球'：7，'一切 星球'：3]，如果我们设置 tfidf = TfidfVectorizer(ngram_range=(1,1))就会有一个小问题，比如“猫吃鱼”和“鱼吃猫”对于 1-gram 来说都是一样的。↵

如果输入的是'Python is useful'，并且将 ngram_range 设置为(1,3)之后可得到'Python' 'is' 'useful' 'Python is' 'is useful' 和'Python is useful'如果是 ngram_range (1,1) 则只能得到单个单词'Python' 'is'和'useful'。在实际使用中，切记不要设置过大的 ngram_range，会给服务器带来很大的性能影响。↵

情感分析

搭建一个情感分析系统。它是文本领域最为经典的项目之一，在各行各业中有着广泛的应用。情感分析问题本身是，给定一个文本并输出它的情感值，情感值无非是正面、负面或者中性。
通过使用一种算法去识别一个文本的情感，这个问题本身属于文本分类问题。 如果只是正面或者负面，就是二分类问题；但如果是正面、负面和中性，则是三分类问题。