其实到上一节，数据挖掘的四种常见问题和算法，我们已经学习完了，从这一小节，我们进入到自然语言处理的相关学习。

我觉得自然语言处理是数据挖掘领域最有意思、最有深度的部分。与我们前面算法所处理的结构化数据不同，自然语言是由人们自由表达的内容，显然是一些非格式化的数据，并且存在着歧义、多义、无序等特点，所以要从这些语言文字中挖掘出有价值的信息也不是一件简单的事情。

但是算法就是如此美妙，能够完成这种看似不可能的任务，虽然目前的效果还不能说尽善尽美，不过随着技术的进步，我们也确实看到这些算法在不断地演化、进步，相信在不久的将来，一定能够达到足够优秀的水准。

自然语言处理的历史

一切技术的发展都是**螺旋式上升**的，自然语言处理技术也依循了这样一个路径。

早期阶段（1956 年以前）：自从提出了著名的 “图灵测试”，人工智能的概念就逐渐进入了研究者的视野。人类的语言，作为人类及其特殊的智慧结晶，让机器理解人类语言当然也是人们迫不及待想要攻克的难题。香农曾提出过概率模型来描述语言，而乔姆斯基提出了基于规则的上下文无关文法。这一阶段还没有太明确的产出，只有一些简单的拼凑。

快速发展（1957-1970）：随后自然语言处理进入了快速发展期，这个阶段两大派别分别从概率模型和规则模型分别进行了深入的研究。同时，一些非常有影响力的概念也在这个时期提了出来，比如神经网络方法和知识库方法（也就是现在知识图谱的原型）。在这个阶段，基于规则的方法占据了主要地位，使用规则构建机器翻译已经小有成效。但是，自然语言处理的难度仍然十分高，众多研究者的热情随着效果的持续不佳逐渐降低了下来。

瓶颈期（1971-1993）：进入 70 年代以后，很多相关的研究都停滞了，不过仍然有小部分研究者在持续进行研究，这个阶段产出的隐马尔科夫模型（HMM）为后面的复苏奠定了基础。

再次爆发（1994 年之后）：到了 90 年代，自然语言处理上的两大困难得到了极大的突破，第一个是随着硬件的升级，运算和存储能力大大增强，原来对硬件设备要求很高的算法可以很快地运行了；第二个是随着互联网的兴起，获取语料变得十分简单，而且对这方面的需求也大大增加，所以自然语言处理再次进入研究的爆发期。尤其是最近几年，深度神经网络不断成熟，自然语言处理技术也随之飞速地发展，各种各样的自然语言处理任务的处理效果都有了很大的提升。

我们这一小节要介绍的 TF-IDF 算法就是自然语言处理早期的产物，而下一小节要介绍的 Word2Vec 算法则是当前最流行的预训练语言模型的前身。好了，接下来让我们进入正题，看一下 TF-IDF 是如何计算的。

算法原理

TF-IDF（Term frequency–inverse document frequency），中文翻译就是**词频 - 逆文档频率**，是一种用来计算**关键词**的传统方法。

在早期处理诸如新闻文本的时候，由于文本的长度一般都很长，要想从内容中获取有价值的信息就需要从中提取关键词，进而使用关键词来表示这篇内容的核心价值。同时，提取关键词也可以看作一种降维的方式，提取完关键词之后，使用关键词再进行文本分类等算法。

那么 TF-IDF 为什么能够较好地提取关键词呢？下面我们就来看一下。

考虑一下，我们是一家新闻机构，每天都有大量的新闻要发布，根据这些内容，我们来看一下如何计算 TF-IDF。

TF（Term Frequency）：TF 的意思就是词频，是指某个词（Term）在一篇新闻中出现的次数。当然，新闻的长度各不相同，为了更加公平，我们可以对 TF 做一下标准化，用词频除以这篇新闻的总词数。

但是，单纯地使用词频来作为关键词的特征明显是不合理的，考虑一些非常常见的词，比如说 “你的”“我的”“今天” 这种词，可能在一篇新闻中会反复出现，但是这种词明显并不重要，于是有了 IDF。

IDF（Inverse Document Frequency）：IDF 称为逆文档频率，这个词我们用公式来看一下可能更容易理解。

$$逆文档频率(IDF) = \log(\frac{新闻的总数量}{包含某个词的新闻数量+1})$$

如果一个词越普通，那它越可能出现在所有的新闻中，那么分母就越大，IDF 的值就越接近 0。

当然，单纯地使用 IDF 作为关键词也不靠谱，如果你自己随便造了一个没人听过的词加在新闻中，比如说 “墽礅”，那么上式的分母非常小，所得到的 IDF 值会非常大，但是这个词明显没有什么意义。

于是，TF-IDF 就是结合了这两个结果，使用 TF×IDF 作为最终的结果，可以看到，TF-IDF 与一个词在新闻中出现的次数成正比，与该词在整个语料上出现的次数成反比。

经过上面这两个值的计算，并把两个值乘起来，TF-IDF 就计算结束了，是不是非常简单明了？接下来我们看一下 TF-IDF 的优缺点。

算法优缺点

非常明显，TF-IDF 的优点就是算法简单，十分容易理解，而且运算速度非常快。

TF-IDF 也有比较明显的缺点，比如在文本比较短的时候几乎无效，如果一篇内容中每个词都只出现了一次，那么用 TF-IDF 很难得到有效的关键词信息；另外 TF-IDF 无法应对一词多义的情况，尤其是博大精深的汉语，对于词的顺序特征也没办法表达。

当然，在传统的基于统计的自然语言处理时代，TF-IDF 仍然是一种十分强大有效的关键词提取方法。

了解了 TF-IDF 算法的基本原理，我们接下来就动动手，用代码来实现 TF-IDF 算法。这次我们使用的是 Gensim 算法工具包，这个工具包我们在前面也介绍过，其中包含了 Word2Vec 等自然语言处理常用的算法工具。同时，里面也内置了一些语料供我们测试使用。

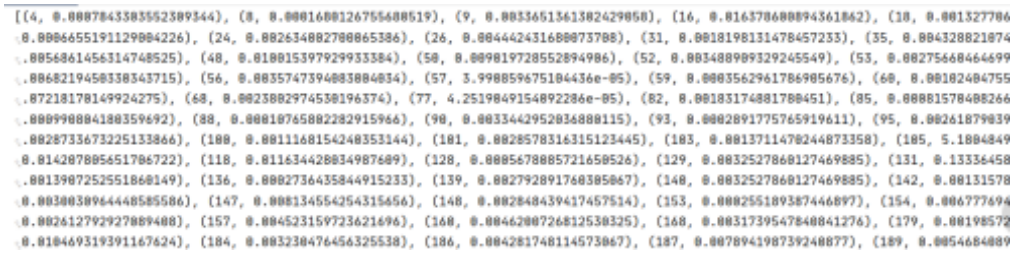
尝试动手

这里我们使用 text8 数据包，是一个来自 Wikipedia 的语料，大小有 30M 多，总共 1701 条数据。第一次使用需要下载，可能要花费一点点时间。

```
import gensim.downloader as api


from gensim.corpora import Dictionary


dataset = api.load("text8")


dct = Dictionary(dataset)


new_corpus = [dct.doc2bow(line) for line in dataset]
```

```
from gensim import models

tfidf = models.TfidfModel(new_corpus)

tfidf.save("tfidf.model")

tfidf = models.TfidfModel.load("tfidf.model")

tfidf_vec = []

for i in range(len(new_corpus)):

    string_tfidf = tfidf[new_corpus[i]]

    tfidf_vec.append(string_tfidf)

print(tfidf_vec)
```

在输出的结果中，一条数据就是一篇内容的所有词的 TF-IDF 值，其中每一个单元有两个值，左边是这个词的 ID，右边是这个词的 TF-IDF 值。细心的你可能会发现，我们输入的内容长度可能会与输出的结果长度不一致，这是因为 Gensim 中的 TF-IDF 算法会过滤停用词、去掉单个的字母等，所以最终结果会缺失一部分。当然，那些词我们认为也不会是关键词，所以在提取关键词的时候也不需要关心。



我们在例子中使用的是英文的语料，如果你想要处理中文，可以安装一个 jieba 分词工具，然后使用 jieba 分词工具把中文语料进行切分，之后的处理步骤就与前面的过程一样了。

```
import jieba

seg_list = jieba.cut("这是一句话，看你切成啥",cut_all=False)

print("Default Mode: " + " ".join(seg_list))
```

在提取关键词方面，单纯地使用 TF-IDF 的效果还不是特别好，可以在 TF-IDF 的基础上，叠加词性、词长度、词位置等特征信息以进行优化。除了 TF-IDF，还有很多关键词提取的方法，如果你对这方面有兴趣，可以在网上进行更深入的查阅。

总结

这一小节，我们开始涉及了一点关于自然语言处理的知识。我在这一小节讲解了一个比较古老，但是很实用的关键词提取算法 TF-IDF，它的原理十分简单、易于理解，通过 TF-IDF 的计算，保留了那些出现频率高的词汇，同时又能够打压那些比较普通的词汇，即便是现在，这个算法仍然有比较广泛的应用。

下一小节，我们将讲一下更加前沿的词嵌入技术 “Word2Vec” 算法。

>