电子商务数据分析

第6.3章 自然语言处理技术：TF-IDF

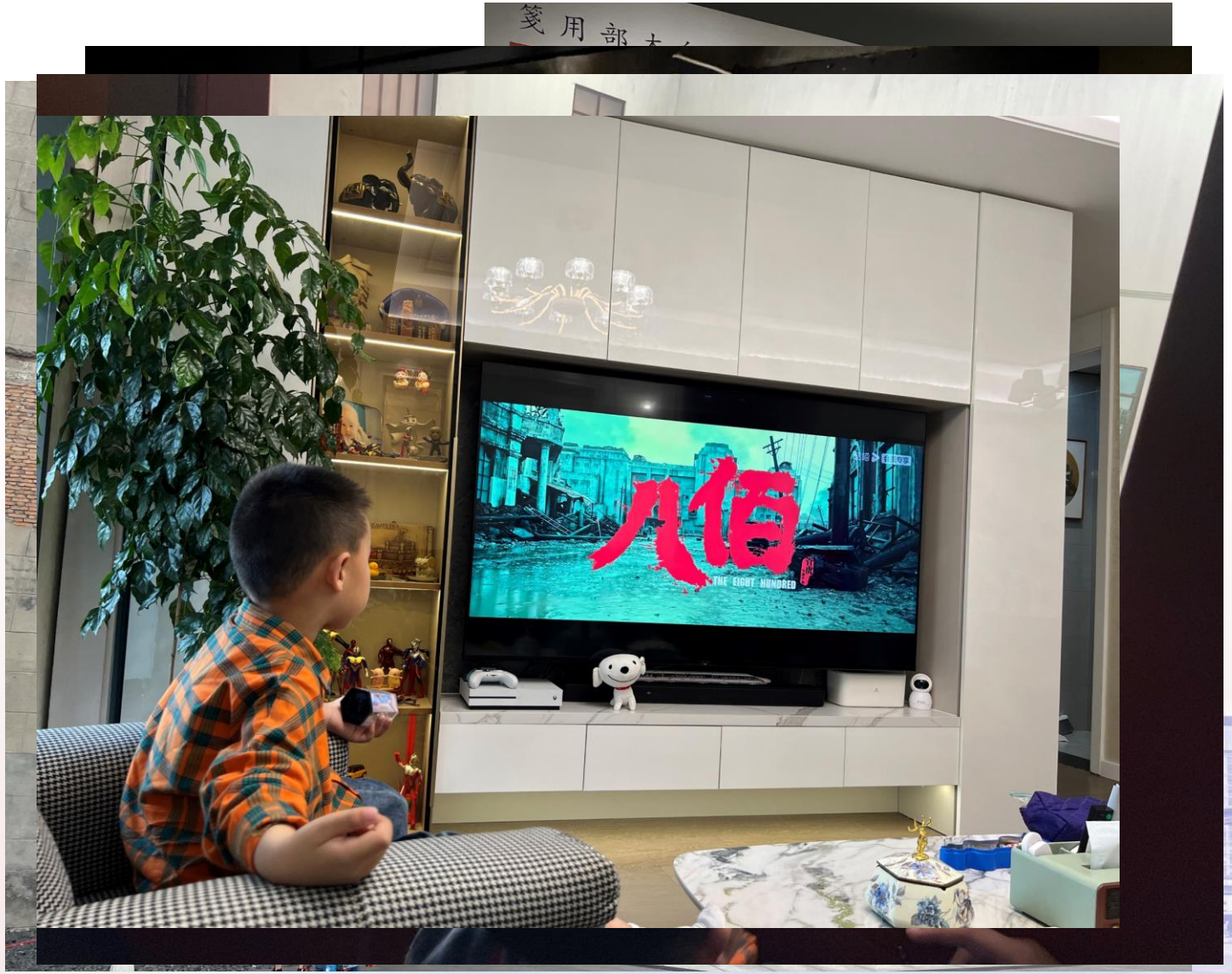
朱桂祥 (9120201070@nufe.edu.cn)
南京财经大学信息工程学院
江苏省电子商务重点实验室
电子商务信息处理国家级国际联合研究中心
电子商务交易技术国家地方联合工程实验室

课程思政专题：四行仓库保卫战

在淞沪会战中，中日双方共有约100万军队投入战斗，战役本身持续了三个月，日军投入9个师团30万人，中国军队投入73个师70余万人。

淞沪会战，使日军被迫转移战略主攻方向，粉碎了日本“三个月灭亡中国”的计划。会战中，中日双方大军拼死搏斗两个多月，到1937年10月底，国军撤离上海。1937年10月26日至30日，国民党政府军第八十八师524团第一营的全体官兵掩护大部队撤退后，奉命进入四行仓库，与日军血战四天四夜，击退敌人多次进攻。它的结束标志着淞沪会战的结束，参加这场保卫战的中国士兵被称为“八百壮士”（实际为450人左右）。

四行仓库保卫战重新振奋了因淞沪会战受挫而下降的中国军民的士气，粉碎了侵华日军三个月灭亡中国的言论。



https://haokan.baidu.com/v?pd=wisenatural&vid=11717294688412779483

Bag-of-words模型

Bag-of-words模型是信息检索领域常用的文档表示方法。在信息检索中，BOW模型假定对于一个**文档**，忽略它的单词顺序和语法、句法等要素，将其**仅仅看作是若干个词汇的集合**，文档中每个单词的出现都是独立的，不依赖于其它单词是否出现。也就是说，文档中任意一个位置出现的任何单词，都不受该文档语意影响而独立选择的。

Bag-of-words模型

例如有如下两个文档：

① Bob likes to play basketball, Jim likes too.

② Bob also likes to play football games.

基于这两个文本文档，构造一个**词表**：

Vocabulary =

{1. “Bob”, 2. “like”, 3. “to”, 4. “play”, 5. “basketball”,6. “also”, 7. “football”, 8. “games”, 9. “Jim”, 10. “too”}。

这个词表一共包含10个不同的单词，利用词表的索引号，上面两个**文档**可以用一个**10维向量表示**(向量中元素为词表中单词在文档中出现的**频率**)：

D=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

① Bob likes to play basketball, Jim likes too.

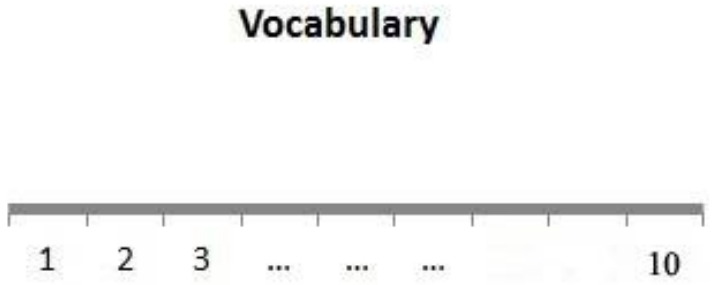
[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]

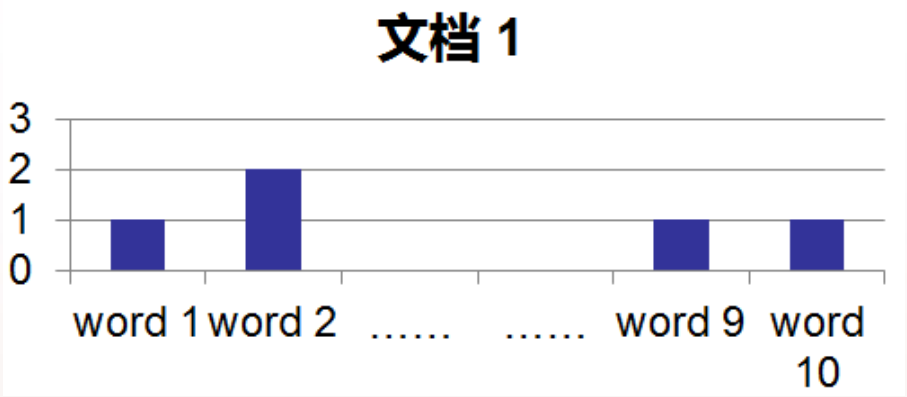
② Bob also likes to play football games.

[1, 1, 1, 1 ,0, 1, 1, 1, 0, 0]

Bag-of-words模型

以上向量也可以用单词的**直方图**表示：词表就相当于直方图的基，新来的要表述的文档向这个基上映射





并不是所有的单词都用来构建词表：

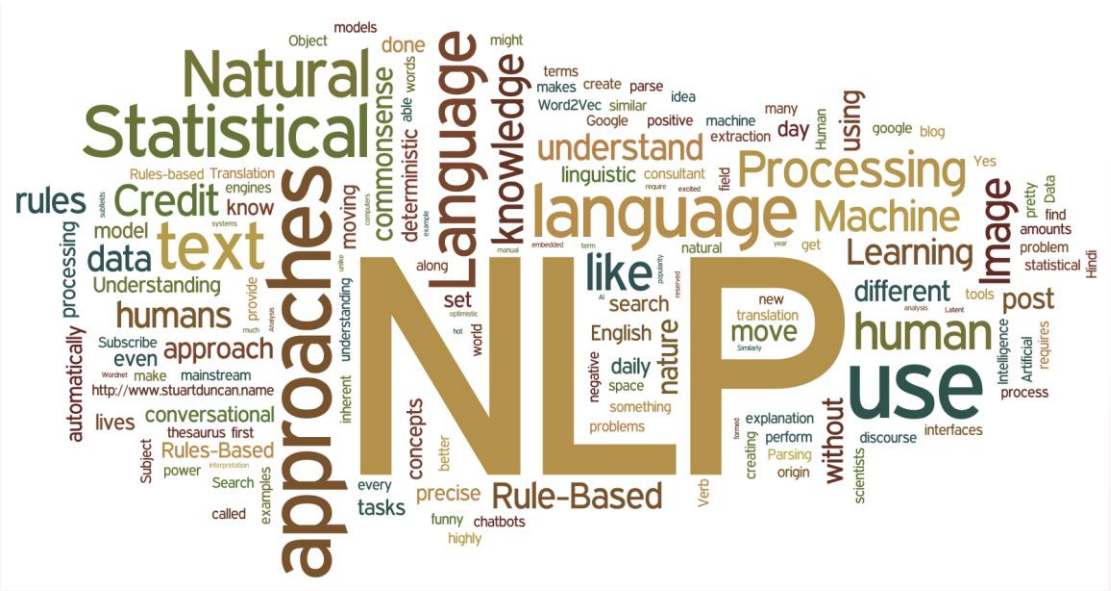
① 相似的词：相似的单词用一个单词表示。例如“walk, walking ,walks”都用“walk”表示。(**聚类问题**)

② 禁用的词：像a, an, the等冠词在各个文档中出现的频率都很高，不容易被区分，这类单词在建立词表的时候不被使用。(**TF-IDF问题**)

Bag-of-words模型

Bag-of-words实现步骤：

1. **词汇表的建立：**大数据聚类，找到合适的聚类中心点——vocabulary
2. **样本的训练：**对每个文档进行训练，得到每个文档的低维表示；
3. **新来样本的识别：**对新来的样本先用词表中单词得到低维表示，然后用**分类器**进行分类预测。

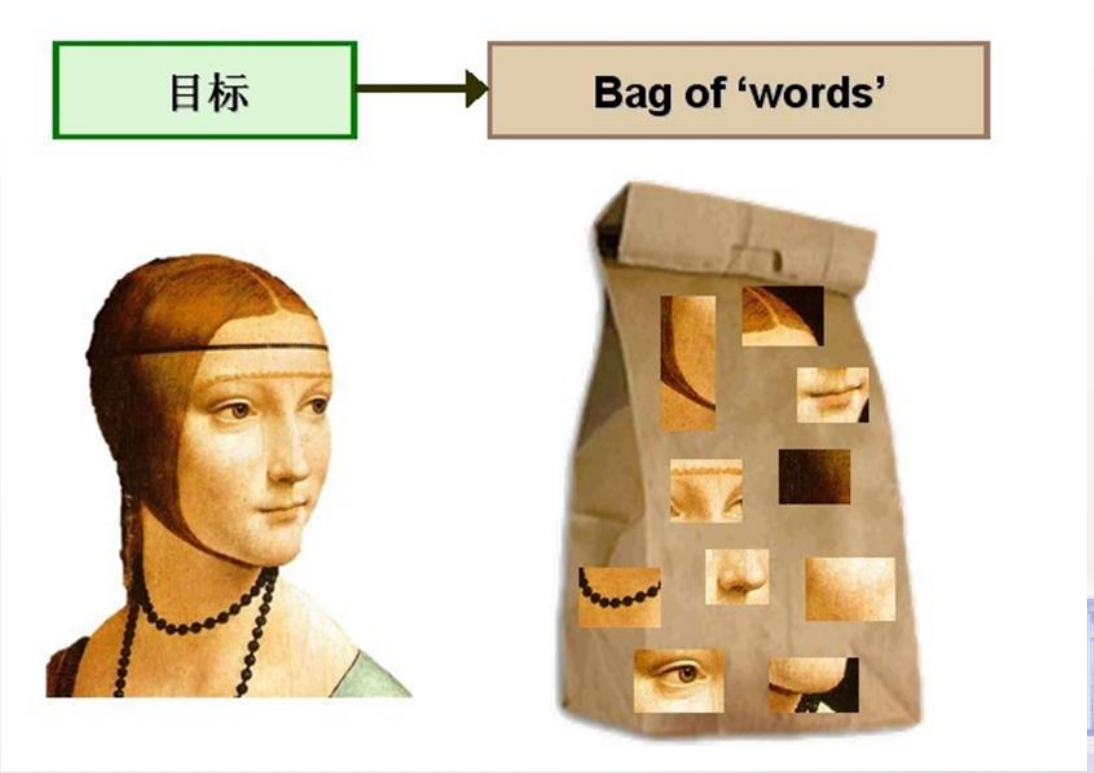


Bag-of-words的应用：

1.假设一个巨大的文档集合，里面一共有M个文档，而文档里面的所有单词提取出来后，一起构成一个包含N个单词的词表，利用Bag-of-words模型，每个文档都可以被表示成为一个N维向量，计算机非常擅长于处理数值向量。这样，就可以利用计算机来完成海量文档的分类过程。

TF-IDF

2.考虑将Bag-of-words模型应用于图像表示，这就是**视觉词袋模型**。为了表示一幅图像，我们可以将图像看作文档，即若干个“视觉单词”的集合，同样的，视觉单词相互之间没有顺序。



TF-IDF

2.考虑将Bag-of-words模型应用于图像表示，这就是**视觉词袋模型**。为了表示一幅图像，我们可以将图像看作文档，即若干个“视觉单词”的集合，同样的，视觉单词相互之间没有顺序。

一个视觉单词

TF-IDF

TF-IDF（Term Frequency–Inverse Document Frequency）是一种用于信息检索与数据挖掘的常用加权技术。TF意思是词频(Term Frequency)，IDF意思是逆文本频率指数(Inverse Document Frequency)。

TF-IDF主要思想：如果某个单词在一篇文章中出现的频率（TF）高，并且在其他文章中很少出现（IDF），则认为这个单词具有很好的类别区分能力，适合用来分类。

TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF加权的各种形式常被搜索引擎应用，作为文件与用户查询之间相关程度的度量或评级。除了TF-IDF以外，因特网上的搜索引擎还会使用基于链接分析的评级方法，以确定文件在搜寻结果中出现的顺序。

TF-IDF

TF-IDF计算公式：

在一份给定的文件里，词频（term frequency，TF）指的是某一个给定的词语在该文件中出现的频率。这个数字是对词数(term count)的归一化，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否。）对于在某一特定文件里的词语ti来说，它的重要性可表示为：

$$tf_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

以上式子中$n_{i,j}$是该词t_i在文件d_j中的出现次数，而分母则是在文件d_j中所有字词的出现次数之和。

TF-IDF

TF-IDF计算公式：

逆向文件频率（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到：

$$idf_i = log \frac{|D|}{|\{j: t_i \in d_j\}|}$$

其中，$|D|$是语料库中的文件总数，$|\{j: t_i \in d_j\}|$代表包含词语t_i的文件数目（即$n_{i,j}$是≠0的文件数目），如果该词语不在语料库中，就会导致被除数为零，因此一般情况下分母使用$1 + |\{j: t_i \in d_j\}|$。

$$tfidf_{i,j} = tf_{ij} \times idf$$

TF-IDF

- 各景点TF-IDF计算结果TOP列表：

  本科毕业设计19级软件工程（嵌入式）朱苑怡：基于机器学习的旅游景点评价分析系统-以南京热门景点为例

景点名称	TF-IDF 计算结果得分较高词汇	
总统府	爆掉	0.860232031
	太美	0.753245876
	花儿	0.707106781
中山陵	绿树成荫	0.809807059
	湿热	0.767168183
	风光秀丽	0.634528877
纪念馆	注意	0.993883735
	强烈建议	0.924616345
	必去	0.797300952
钟山-明孝陵	有意思	0.796720904
	著名	0.7829808
	文化遗产	0.707106781

TF-IDF

TF-IDF案例：

本科毕业设计19级软件工程（嵌入式）朱苑怡：基于机器学习的旅游景点评价分析系统-以南京热门景点为例

夫子庙-秦淮河	商业	0.814305655
	氛围	0.768082747
	惹人醉	0.722408805
南京博物院	值得	0.869583783
	推荐	0.67694453
	性价比	0.661328779
玄武湖	普通	0.877136839
	赏花	0.648195271
	漂亮	0.632624044
鸡鸣寺	南京城	0.639606242
	古刹	0.625201258
	美好	0.597291148

TF-IDF

TF-IDF案例：

- 融合TF-IDF的评论关键词云图

(a) 总统府

(b) 中山陵

(c) 遇难同胞纪念馆

(d) 钟山-明孝陵

(e) 夫子庙-秦淮河

(f) 南京博物院

(g) 玄武湖

(h) 鸡鸣寺

TF-IDF

TF-IDF Python实现：

Natural Language Toolkit，自然语言处理工具包，在NLP领域中，最常使用的一个Python库。

```
from nltk.text import TextCollection
from nltk.tokenize import word_tokenize

#首先，构建语料库corpus
sents=['this is sentence one','this is sentence two','this is sentence three']
sents=[word_tokenize(sent) for sent in sents] #对每个句子进行分词
print(sents)  #输出分词后的结果
corpus=TextCollection(sents)  #构建语料库
print(corpus)  #输出语料库

#计算语料库中"one"的tf值
tf=corpus.tf('one',corpus)     # 1/12
print(tf)

#计算语料库中"one"的idf值
idf=corpus.idf('one')       #log(3/1)
print(idf)

#计算语料库中"one"的tf-idf值
tf_idf=corpus.tf_idf('one',corpus)
print(tf_idf)
```

```
[['this', 'is', 'sentence', 'one'], ['this', 'is', 'sentence', 'two'], ['this', 'is', 'sentence', 'three']]
<Text: this is sentence one this is sentence two...>
0.08333333333333333
1.0986122886681098
0.0915510240556758
```

谢谢观赏 下节课见