第三讲 - 文本统计量和向量化表示

张建章

阿里巴巴商学院
杭州师范大学

2023-02-22

目录

语料库构建：将原始文本预处理后，按照结构化的方式组织和保存。

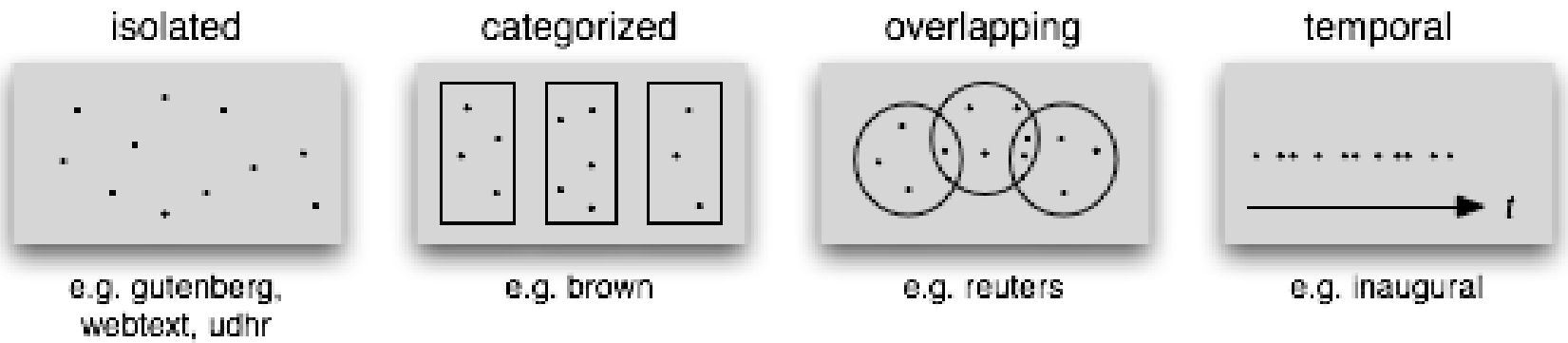


图 1: 常见语料库组织结构

语料库管理：从多种视图高效检索语料库，检索结果可进一步用于文本统计分析和向量化。

本讲以中文新闻作为示例语料讲解语料库的构建和管理，并在此基础上学习常见的文本统计量和文本向量化表示方法。该语料来自搜狗实验室发布的新闻分类数据集。

新闻语料预处理

主要包括：分句、分词、词性标注。使用 `spaCy` (高效、开源、维护更新及时、支持多种语言和多种 NLP 任务，支持加载调用已训练好的模型和训练自己的模型) 进行预处理，示例用法如下：

```
import spacy
nlp = spacy.load("zh_core_web_sm")

text = '''2023年短道速滑世锦赛落幕，
↪  在最后一个项目男子5000米接力决赛中，中国队夺得金牌。
↪  这是本届世锦赛中国队夺得的首金。'''

doc = nlp(text)
for sent in doc.sents:
    for token in sent:
        print(token.text, token.pos_)
    print('-'*20)
```

定义一个文本预处理函数，应用到每条新闻文本上：

```
def tagging(text):
    f_list = []
    paragraphs = html.unescape(text).strip().split('\n')
    for paragraph in paragraphs:
        if paragraph.strip():
            p_list = []
            doc = nlp(paragraph.strip())
            for sent in doc.sents:
                t_list = []
                for token in sent:
                    t_list.append(token.text+'/'+token.pos_)
                p_list.append(' '.join(t_list))
            f_list.append('\n'.join(p_list))
    tagged_text = '\n\n'.join(f_list)
    return tagged_text
```

对所有新闻文本进行预处理，并保存预处理结果：

```
import os
raw_data_dir = './cn_news/'
tagged_data_dir = './cn_news_tagged/'
categories = os.listdir(raw_data_dir)
for category in categories:
    files = os.listdir(raw_data_dir + category)
    for file in files:
        file_path = raw_data_dir + category + '/' + file
        if not os.path.exists(tagged_data_dir + category):
            os.mkdir(tagged_data_dir + category)
        new_file_path = tagged_data_dir + category + '/' + file
        with open(file_path, encoding='gbk',errors='ignore') as
        ↪  f:
            raw_text = f.read()
        tagged_text = tagging(raw_text)
        with open(new_file_path,'w') as f:
            f.write(tagged_text)
```

构建新闻分类标注语料库

使用 NLTK 提供的语料库 API，从前述的新闻标注结果文件构建分类标注语料库。

```
from nltk.corpus.reader import CategorizedTaggedCorpusReader

creader = CategorizedTaggedCorpusReader('./cn_news_tagged/',
↪  '.*',cat_pattern = r'(.+)/.+txt')
```

与文件读写和字符串操作的简单组合相比，NLTK 提供的语料库构建 API 效率更高，支持多种语料库检索和查询操作。

语料库管理

NLTK 提供的常用的语料库管理功能如下：

Example	Description
`fileids()`	the files of the corpus
`fileids([categories])`	the files of the corpus corresponding to these categories
`categories()`	the categories of the corpus
`categories([fileids])`	the categories of the corpus corresponding to these files
`raw()`	the raw content of the corpus
`raw(fileids=[f1,f2,f3])`	the raw content of the specified files
`raw(categories=[c1,c2])`	the raw content of the specified categories
`words()`	the words of the whole corpus
`words(fileids=[f1,f2,f3])`	the words of the specified fileids
`words(categories=[c1,c2])`	the words of the specified categories
`sents()`	the sentences of the whole corpus
`sents(fileids=[f1,f2,f3])`	the sentences of the specified fileids
`sents(categories=[c1,c2])`	the sentences of the specified categories
`abspath(fileid)`	the location of the given file on disk
`encoding(fileid)`	the encoding of the file (if known)
`open(fileid)`	open a stream for reading the given corpus file
`root`	if the path to the root of locally installed corpus
`readme()`	the contents of the README file of the corpus

图 2: NLTK 中常用的语料库管理方法

语料库检索

上下文是确定词语语义 (语义消歧) 的关键信息，也可用于确定两个词语的相似度 (思考英语阅读理解中你对陌生/罕见词含义的推断过程)。下面代码使用 NLTK 中的语料库方法实现上述两个功能。

```
# 通过上下文理解单词monstrous的意思
# text1为小说白鲸记文本
from nltk.book import *
text1.concordance("monstrous")

# of Whale - Bones ; for Whales of a monstrous size are
↪  oftentimes cast up dead u
# ll over with a heathenish array of monstrous clubs and spears
↪  . Some were thick
# ere to enter upon those still more monstrous stories of them
↪  which are to be fo

# 鲸鱼骨头; 因为<巨大的>鲸鱼经常被抛出死去
# 到处都是异教徒的<怪异的>棍棒和长矛。有些很厚
# 在进入他们将要讲述的那些更<可怕的>故事之前
```

语料库检索

```
# text1为小说白鲸记文本
# text2为小说理智与情感文本
from nltk.book import *
text1.similar("monstrous")
# mean part maddens doleful gamesome subtly uncommon careful
↪  untoward exasperate loving passing mouldy christian few true
↪  mystifying imperial modifies contemptible
text2.similar("monstrous")
# very heartily so exceedingly remarkably as vast a great
↪  amazingly extremely good sweet

# 寻找多个词语的公共上下文, a very/monstrous lucky
text2.common_contexts(["monstrous", "very"])
a_pretty is_pretty am_glad be_glad a_lucky
```

在小说 < 大白鲸 > 和 < 理智与爱情 > 中单词 *monstrous* 的含义明显不同，在 < 理智与爱情 > 中，其含义偏积极，有时还会像 *very* 一样作为语气增强词使用。

文本长度和词表规模

通过计算文本中词汇总数 (N) 和词表规模 (V)，可以计算不同类型文本的词汇丰富度。

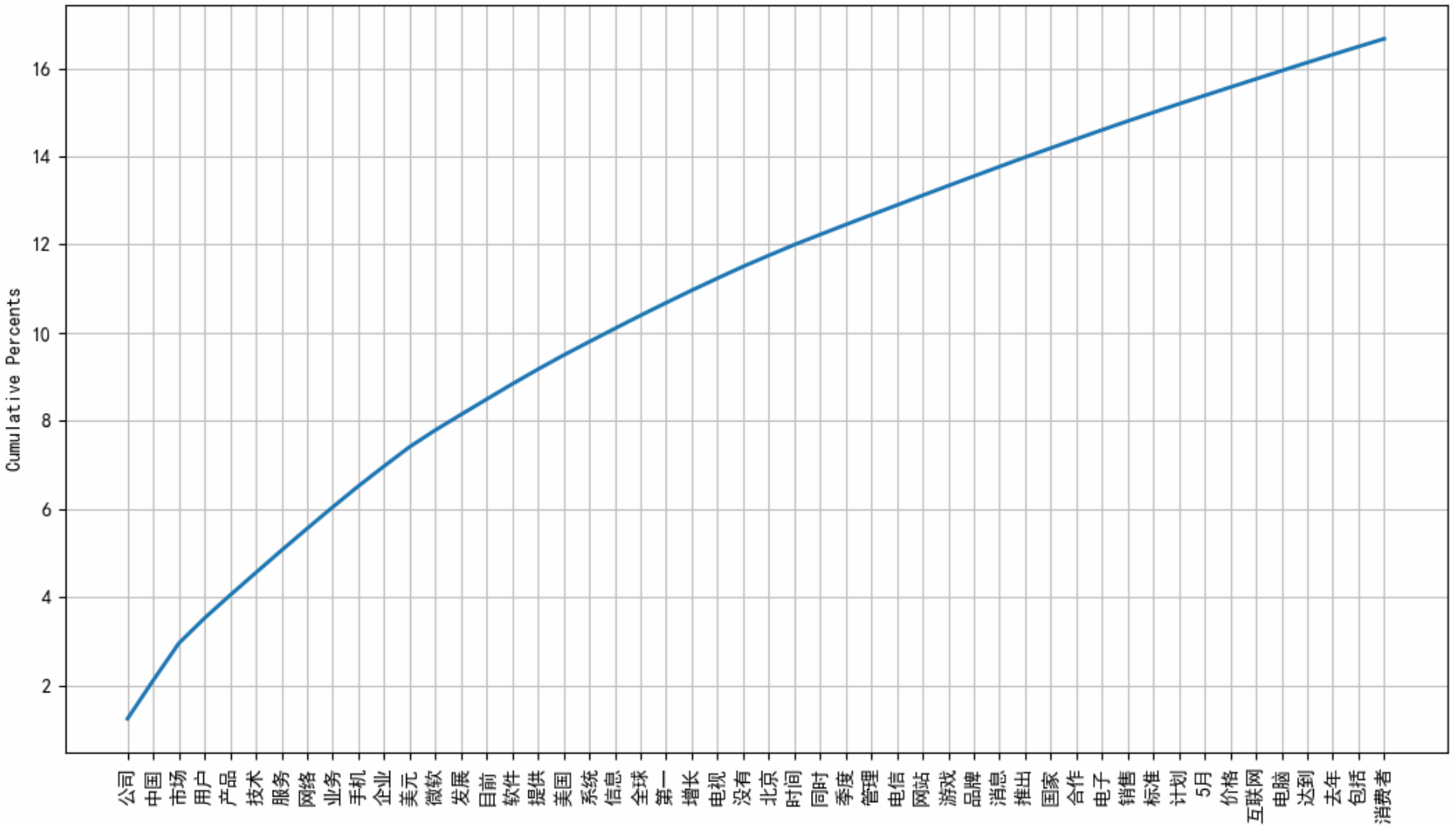
$$lexical\_richness = \frac{V}{N}$$

下面以中文新闻语料库为例，计算不同类别的词汇丰富度。

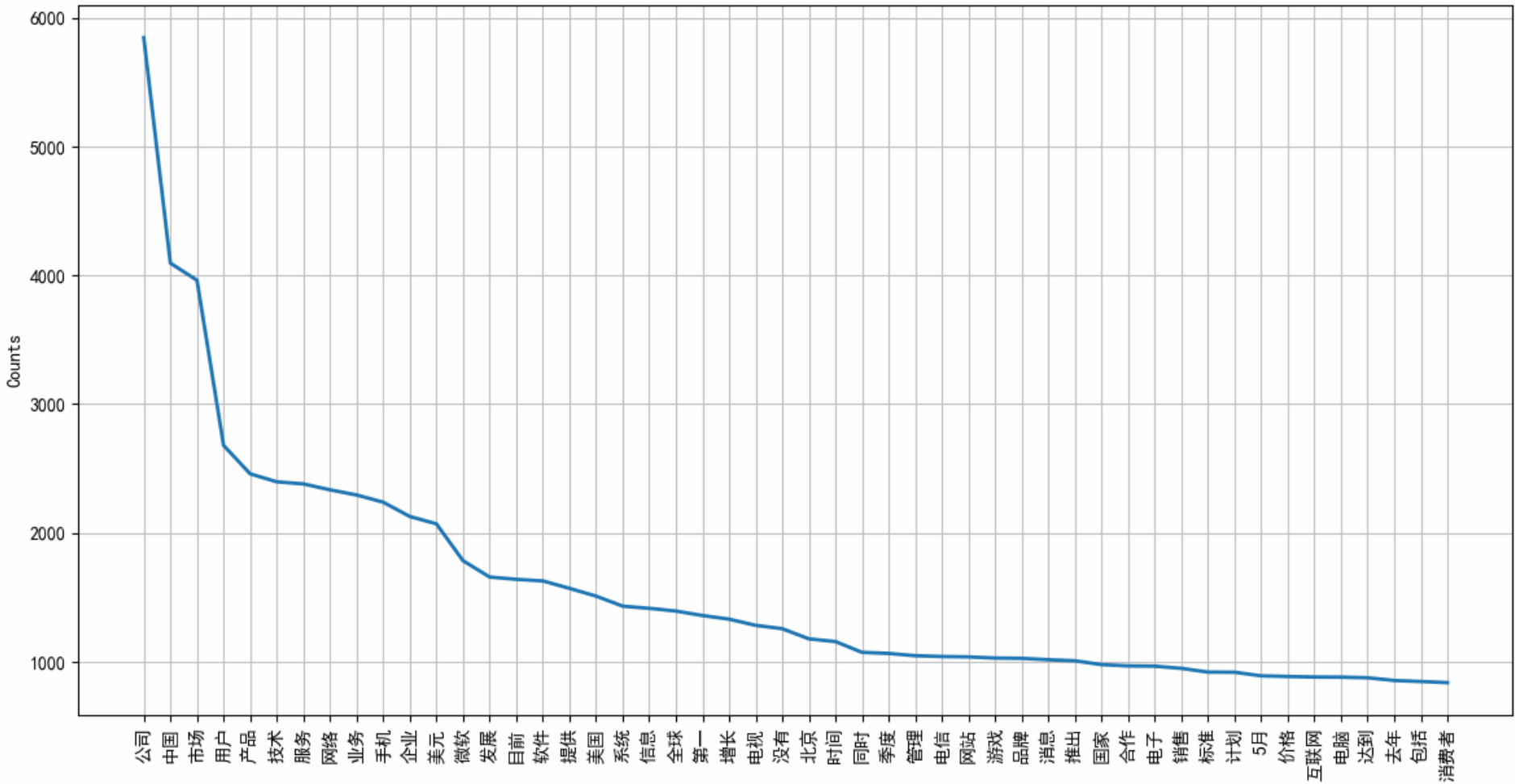
```
from nltk.corpus.reader import CategorizedTaggedCorpusReader

creader = CategorizedTaggedCorpusReader('./cn_news_tagged/',
↪  '.*',cat_pattern = r'(.+)/.+txt')
it_words = creader.tagged_words(categories=['IT'])
cult_words = creader.tagged_words(categories=['Culture'])
print(len(it_words), len(cult_words))
print(len(set(it_words)), len(set(cult_words)))
def lexical_diversity(text):
  return len(set(text)) / len(text)
print(lexical_diversity(it_words), lexical_diversity(cult_words))
```

词语频率分布

计算词表中词语的出现频率 (Top 50)，绘制词语累积频率分布图。



计算词表中词语的出现频数 (Top 50)，绘制词语频数分布图 (长尾分布)，只有少量高频词，大多数词语出现的频率都很低 (词频曲线陡降并趋于平缓)。



词语频率分布

识别文本中的高频词语，并绘制词云 (Top 50)。

条件频率统计

统计词汇在给定条件下的频率，如统计某一词语在不同类别的新闻中出现的频率，某一词语在不同年份语料中出现的频率等。可以使用 `NLTK` 中的条件频率统计方法 `ConditionalFreqDist` 进行计算和绘图，将条件和词语表示为二元组形式，例如，(文本类别, 词语)。得分，货币，心情三个词语在不同类别新闻语料中的频率如下表。

	得分	货币	心情
Culture	9	62	143
Education	99	79	148
Finance	0	256	7
Health	6	0	192
IT	31	9	12
Military	1	5	28
Recruit	31	26	249
Sports	704	0	71
Travel	0	59	62

从具有时间跨度的语料库中按照时间维度统计词频，可以挖掘词语含义、使用模式随时间变化的趋势。下图显示了在美国总统就职演说语料库中，以 *america*-和 *citizen*-为前缀的单词的使用频率随时间变化的趋势。code

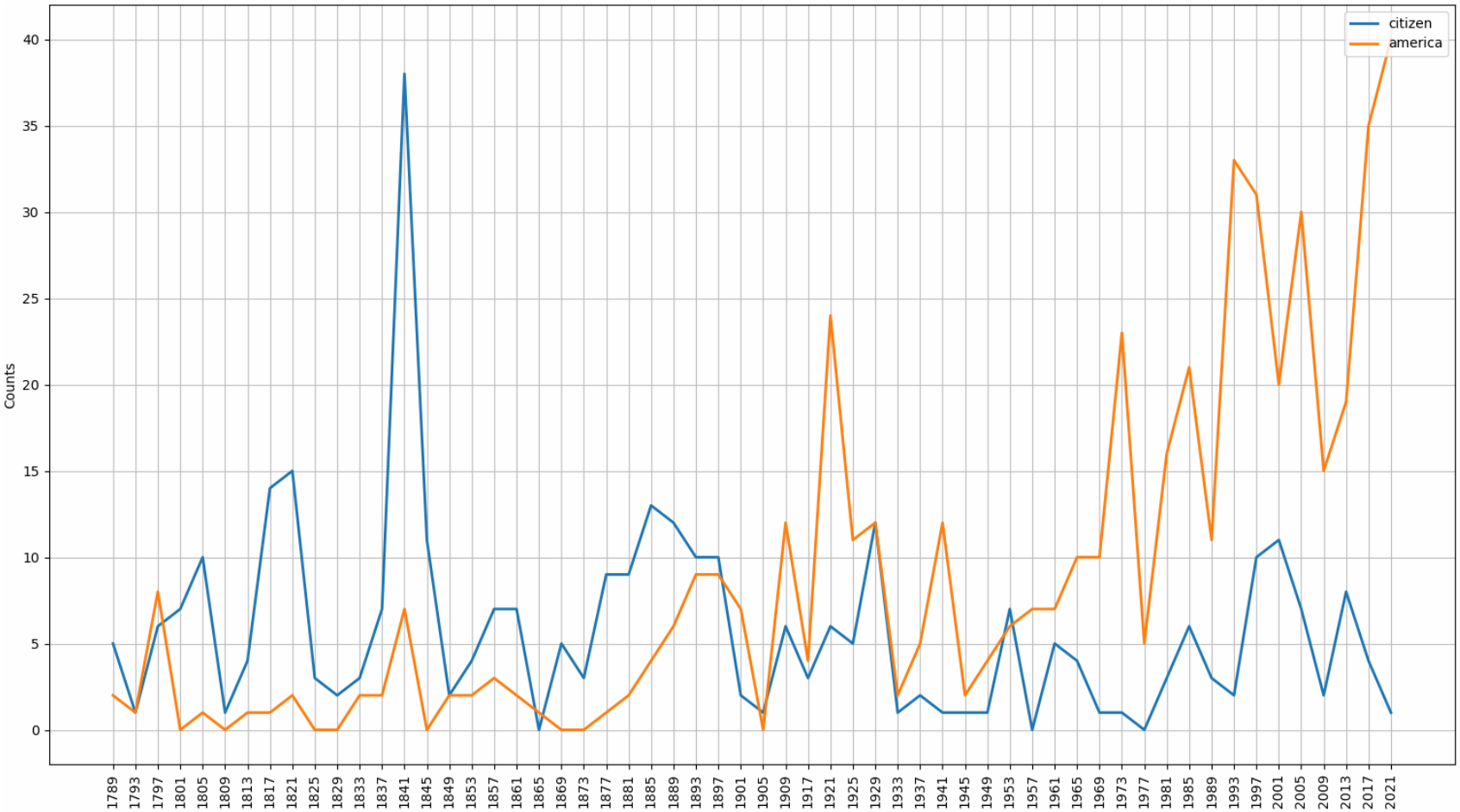


图 3: NLTK 中常用的语料库管理方法

词语搭配挖掘

通过统计两个或多个词语在语料库中的共现频率，挖掘词语搭配 (collocation)，可应用于术语挖掘、新词 (短语) 发现等场景。

常见的度量词语之间关联性的指标有 t 统计量、卡方统计量、似然比、点互信息。计算这些指标均需要使用如下图所示的词频统列联表。

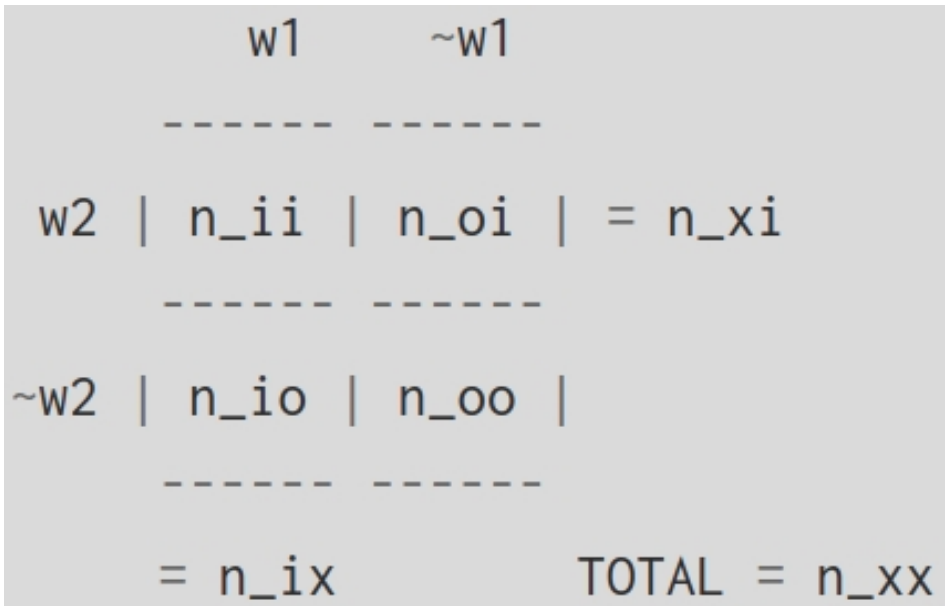


图 4: 词频统列联表

《*Foundations of Statistical Natural Language Processing*》第五章

从信息论角度度量两个词语相关性的点互信息指标定义如下：

$$PMI(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

假定一个语料库中总规模为 14,307,668。单词 *Ayatollah* 和单词 *Ruhollah* 的各自出现频率和两者共现 (考虑词序) 频率分别为 42, 20, 20，使用极大似然估计计算概率，则词语搭配 *Ayatollah Ruhollah* 的点互信息值计算如下：

$$PMI(Ayatollah, Ruhollah) = \log_2 \frac{\frac{20}{14307668}}{\frac{42}{14307668} \times \frac{20}{14307668}} \approx 18.38$$

应用 NLTK 从中文 IT 新闻语料中挖掘词语搭配示例如下：

```
import nltk
from nltk.collocations import *
from nltk.corpus.reader import CategorizedTaggedCorpusReader
bigram_measures = nltk.collocations.BigramAssocMeasures()
creader = CategorizedTaggedCorpusReader('./cn_news_tagged/',
↪  '.*',cat_pattern = r'(.+)/.+txt')
IT_t = nltk.Text(creader.words(categories=['IT']))
finder = BigramCollocationFinder.from_words(IT_t.tokens,
↪  window_size=5)
finder.apply_freq_filter(2)
finder.apply_word_filter(lambda w: len(w) < 3)
finder.nbest(bigram_measures.pmi, 10) # 点互信息
finder.nbest(bigram_measures.chi_sq, 10) # 卡方
finder.nbest(bigram_measures.likelihood_ratio, 10) # 似然比
finder.nbest(bigram_measures.student_t, 10) # t统计量
```

```
[('PDP', '5017'),
 ('DFC', '戴卫·考尔'),
 ('BELLVILLES', 'TEESZA'),
 ('(Location', 'Free'),
 ('(Location', 'LFTV)'),
 ('(Luxe', 'Interactive)'),
 ('(Wired', 'Transmission'),
 ('01.jpg', 'target'),
 ('0800', '918-066'),
 ('1003', '95万')]
```

点互信息

```
[('PDP', '5017'),
 ('Ready', 'Drive'),
 ('DFC', '戴卫·考尔'),
 ('BELLVILLES', 'TEESZA'),
 ('Julie', 'Ryan'),
 ('(Location', 'Free'),
 ('(Location', 'LFTV)'),
 ('(Luxe', 'Interactive)'),
 ('(Wired', 'Transmission'),
 ('01.jpg', 'target')]
```

卡方统计量

```
[('Xbox', '360'),
 ('CNET', '资讯网'),
 ('Play', 'Station'),
 ('Windows', 'Live'),
 ('Core', 'Duo'),
 ('Windows', 'Vista'),
 ('巴塞罗那', '俱乐部'),
 ('Windows', 'Mobile'),
 ('Web', '2.0'),
 ('PS3', '游戏机')]
```

似然比

```
[('Xbox', '360'),
 ('Windows', 'Live'),
 ('CNET', '资讯网'),
 ('Play', 'Station'),
 ('Windows', 'Vista'),
 ('Windows', 'Mobile'),
 ('Core', 'Duo'),
 ('360', '游戏机'),
 ('PS3', '游戏机'),
 ('Xbox', '游戏机')]
```

t统计量

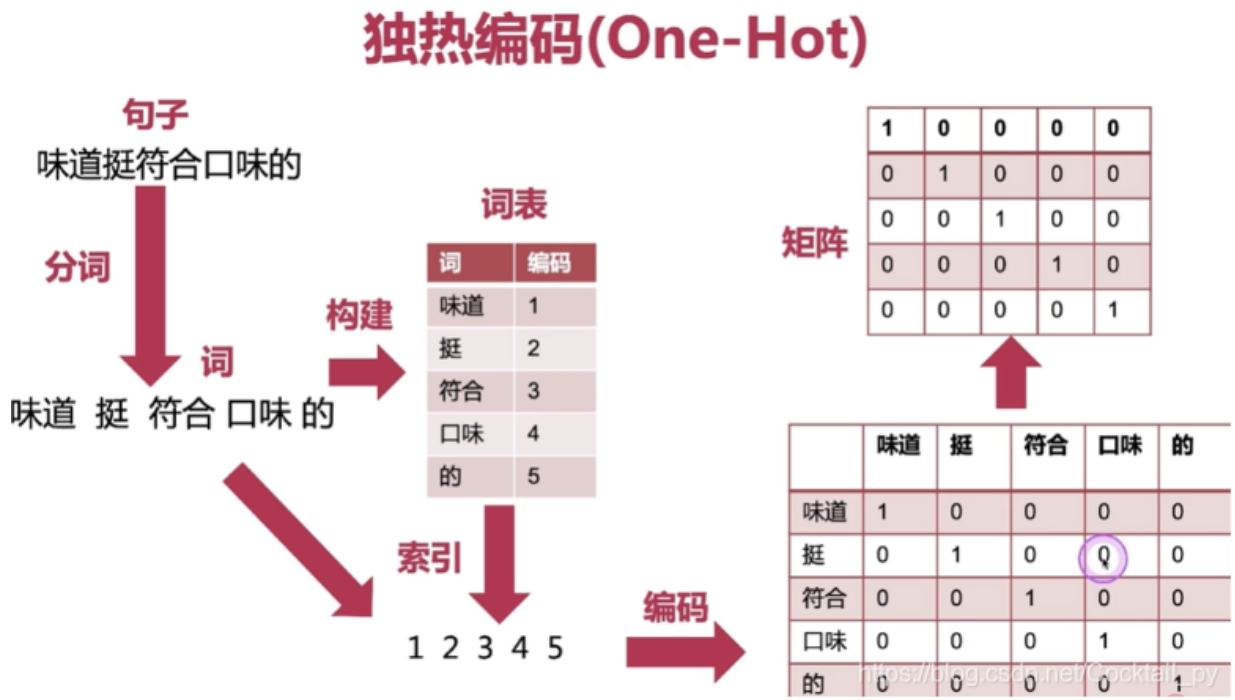
上例中，似然率和 t 统计量的挖掘结果明显好于卡方统计量和点互信息法，即，挖掘出的搭配更有意义。

独热向量编码

使用一个长度为 N 的向量表示一个词语，N 表示词表的规模，i 表示词语 w_i 在词表中的索引值，则词语 w_i 的独热向量表示如下：

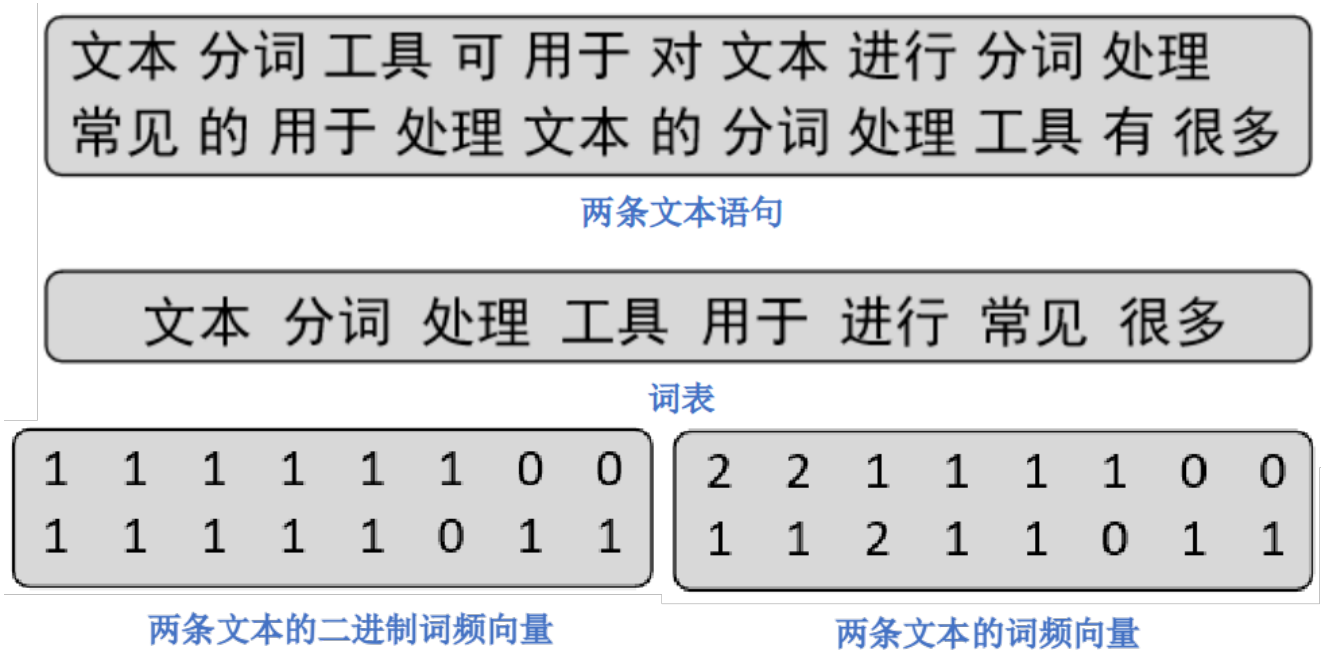
$$v_i = [0, 0, ..., 1, 0, 0, ...]$$

向量 v_i 中第 i 个元素为 1，其余元素均为 0。



词频向量

将句子中的词语的独热向量求和即可得到句子的词频向量表示。



基于简单词频统计的文本向量化表示方法不考虑文本中词语的顺序和词语所处的上下文，将文本视为一个袋子，里面装着构成该文本的词语，因此，这类模型又称为词袋模型 (bag of words)。

词频-逆文档频向量

之前方法只考虑词语在文档中的出现频率，进一步考虑词语在整个语料库中出现频率，可以更全面衡量词语对于文档的重要性。文本挖掘中常用的词频-逆文档频 (term frequency-inverse document frequency, TF-IDF) 指标便是基于上述思想设计。

$$\mathrm{TF}(t, d) = \frac{\mathrm{count}(t, d)}{\sum_{w \in d} \mathrm{count}(w, d)}$$

$$\mathrm{IDF}(t, D) = \log \frac{\text{total documents in } D}{\text{number of documents containing term } t}$$

$$\text{TF-IDF}(t, d, D) = \mathrm{TF}(t, d) \cdot \mathrm{IDF}(t, D)$$

其中，t 表示词汇，d 表示文档，D 表示文档集合，IDF 计算中通常使用自然对数，即，底数为 e。

文档

今天/天气/晴朗。
太阳/今天/格外/明亮。
今天/太阳/在/晴朗/的/天空/中/格外/明亮。

分别计算 TF、IDF

$$\text{TF}(\text{太阳},\ 1) = \frac{0}{4} = 0$$
$$\text{TF}(\text{太阳},\ 2) = \frac{1}{5} = 0.2$$
$$\text{TF}(\text{太阳},\ 3) = \frac{1}{9} \approx 0.11$$
$$\text{IDF}(\text{太阳}, D) = \log \frac{3}{2} \approx 0.41$$

计算 TF-IDF

$$\text{TF-IDF}(\text{太阳},\ 1, D) = 0 * 0.41 = 0$$
$$\text{TF-IDF}(\text{太阳},\ 2, D) = 0.2 * 0.41 \approx 0.082$$
$$\text{TF-IDF}(\text{太阳},\ 3, D) = 0.11 * 0.41 \approx 0.045$$

词频-逆文档频向量

```
import numpy as np
from nltk.corpus.reader import CategorizedTaggedCorpusReader
from sklearn.feature_extraction.text import CountVectorizer,
↪  TfidfTransformer

tokenized_docs = []
for file_id in creader.fileids(categories=['IT']):
  doc = creader.words(fileids=[file_id])
  clean_doc = filter(cn_filter, doc)
  tokenized_docs.append(clean_doc)

count_vectorizer = CountVectorizer(tokenizer=lambda
↪  text:list(text), lowercase=False, min_df = 5)

count_matrix = count_vectorizer.fit_transform(tokenized_docs)
binary_matrix = np.where(count_matrix.toarray() > 0, 1, 0)
tfidf_transformer = TfidfTransformer()
tfidf_matrix = tfidf_transformer.fit_transform(count_matrix)
```

词嵌入向量 V.S. 独热向量

独热向量维度高、稀疏，即，向量维度等于词表规模，向量中的只有一个非零元素，但是每个维度的值具有可解释性，即，每个维度对应词表中的一个词语。

词嵌入 (word embeddings) 向量维度低、稠密，即，向量维度相比词表规模要小很多，通常低于 1000 维，向量中的元素都为浮点数，但是每个维度的值不具有可解释性。

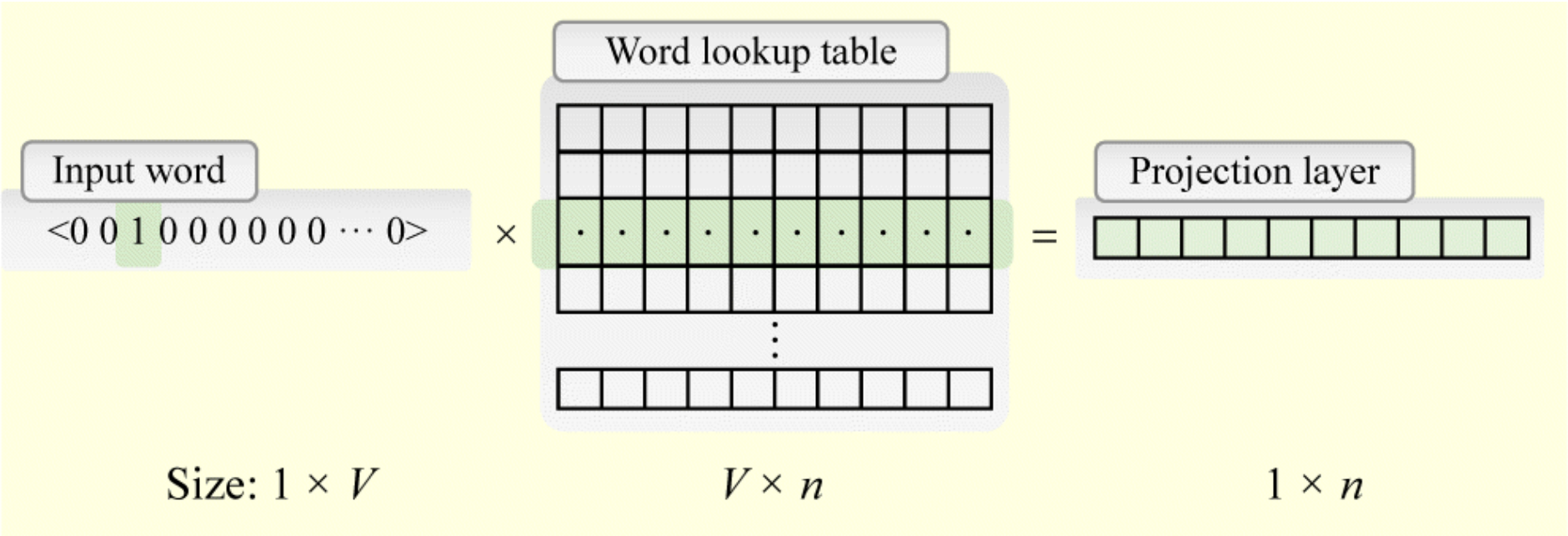
对于 IT 新闻语料中的 电脑 一词，其词嵌入向量和独热编码向量示例如下：

词嵌入和词频向量示例

$$[0.107, -0.201, 0.123, ..., -0.248, -0.062]_{300}$$
$$[0, 0, 1, 0, ..., 0]_{7045}$$

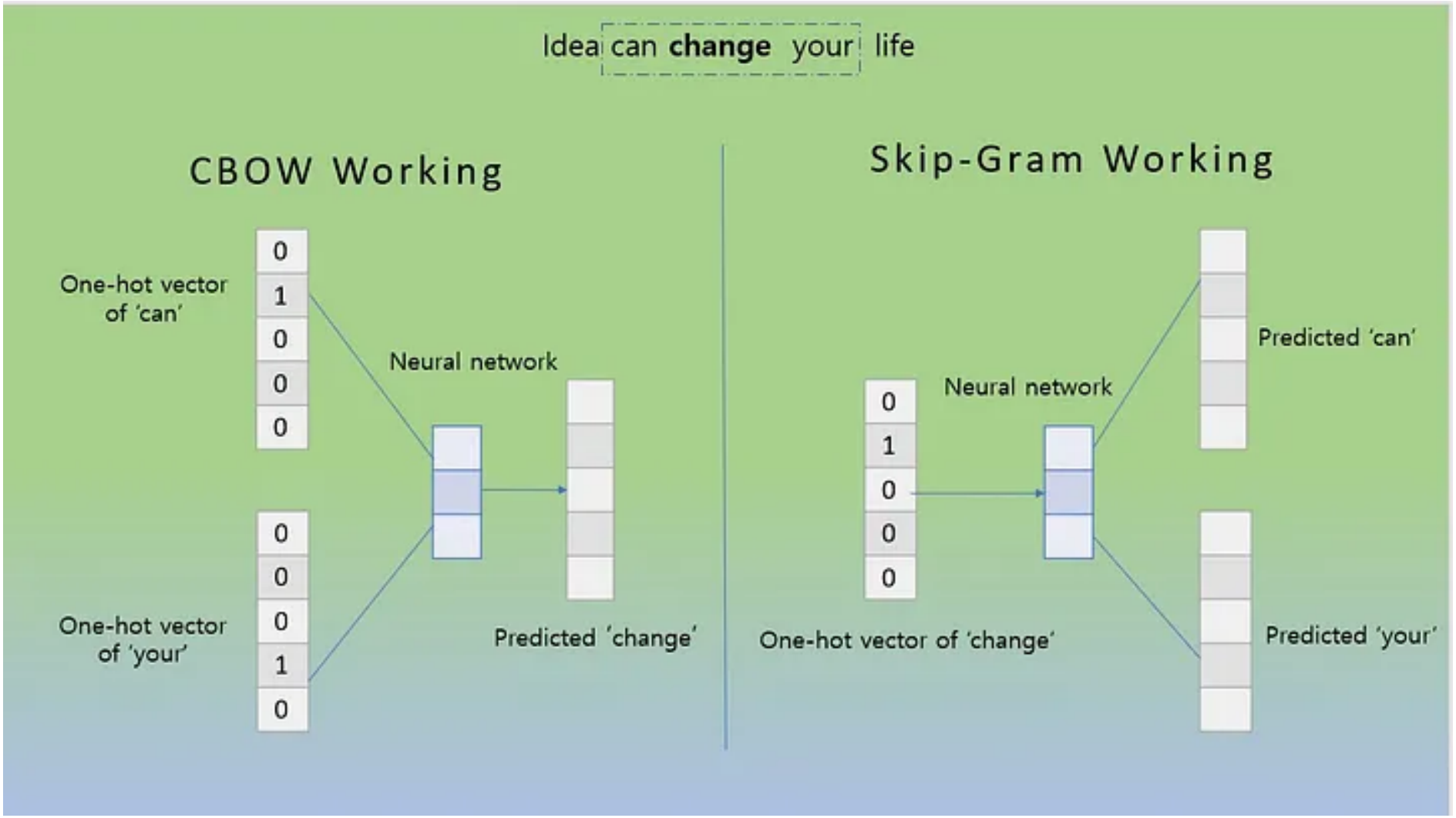
上例中，词嵌入向量的维度为 300，IT 新闻语料的词表包含 7045 个词。

词嵌入向量是基于深度神经网络的自然语言处理模型的基本输入，通常将保存词向量的矩阵称为词嵌入查找表 (embedding lookup table)。将词语的独热向量与词嵌入矩阵相乘即可得到词语的嵌入向量表示。

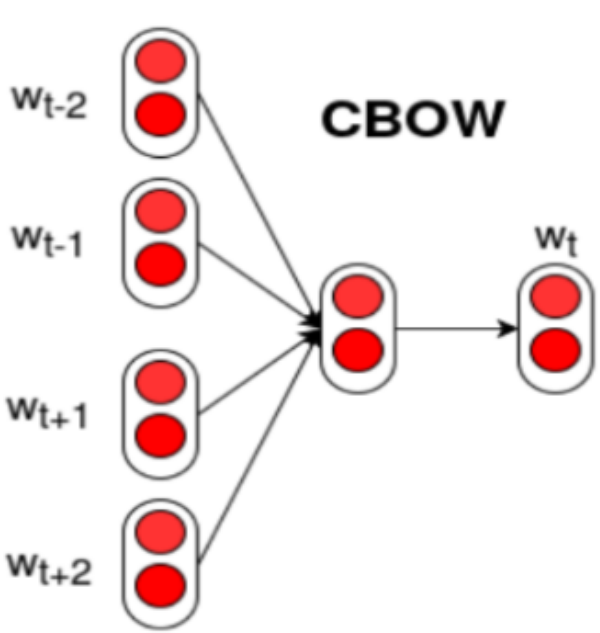


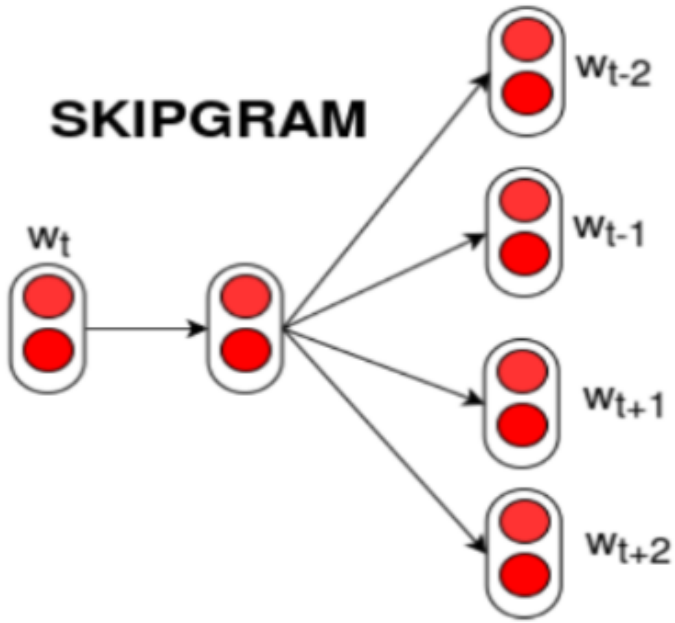
上图中，V 表示词表规模，n 表示词嵌入向量的维度。矩阵中每一行向量对应一个词语的嵌入表示。

Word2Vec 由谷歌研究团队于 2013 年提出，其核心思想是通过单词在文本中的上下文来学习它们的向量表示，学习方法为神经网络模型。它有两种主要的学习架构：Skip-Gram 和 Continuous Bag of Words (CBOW)。



Word2Vec 两种学习架构的目标函数如下，即，使用极大似然估计，最大化对数似然。





$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_t|w_{t+j})$$

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

对上述目标函数取负，即得到损失函数，训练目标为最小化损失。

以 Skip-Gram 架构为例，目标函数推导如下：

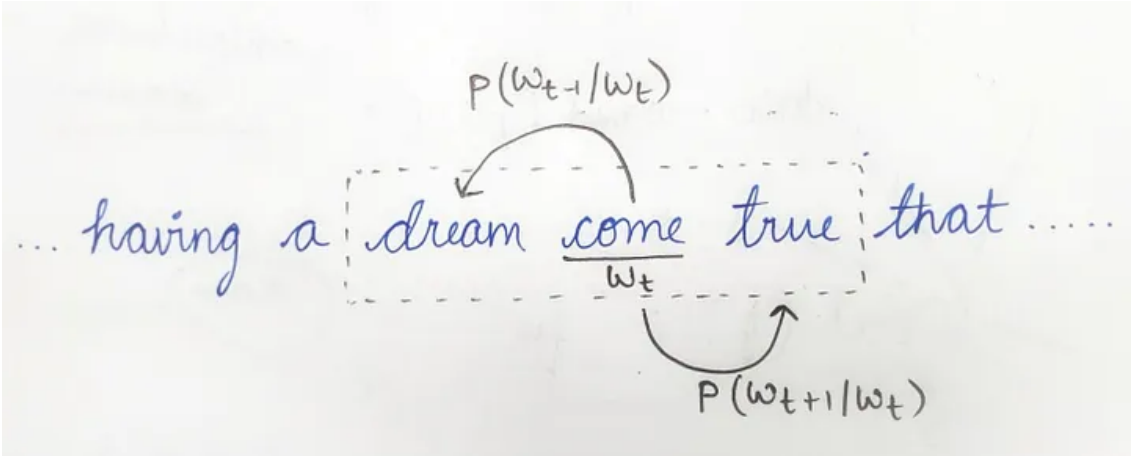


图 5: 使用中心词预测上下文，窗口为 1

对于一个大小为 T 的训练语料库，给定中心词 w_t，针对窗口大小为 c 的上下文中的每个词语进行预测，似然值如下：

$$P_{\text{Skip-Gram}} = \prod_{t=1}^{T} \prod_{-c \le j \le c, j \neq 0} p(w_{t+j} | w_t; \boldsymbol{\theta})$$

其中，$\boldsymbol{\theta}$ 为训练过程中需要学习的参数值，包括词语的向量表示。

对数似然值：

$$L_{\text{Skip-Gram}} = \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{t+j} | w_t; \boldsymbol{\theta})$$

损失函数，即负的平均对数似然值：

$$-\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{t+j} | w_t; \boldsymbol{\theta})$$

因此，Skip-Gram 的优化目标为：

$$\min_{\boldsymbol{\theta}} -\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{t+j} | w_t; \boldsymbol{\theta})$$

其中 $p(w_{t+j}|w_t)$ 通过 $Softmax$ 函数计算：

$$p(w_{t+j}|w_t) = \frac{\exp(v'_{w_{t+j}} \cdot v_{w_t})}{\sum_{i=1}^{V} \exp(v'_i \cdot v_{w_t})}$$

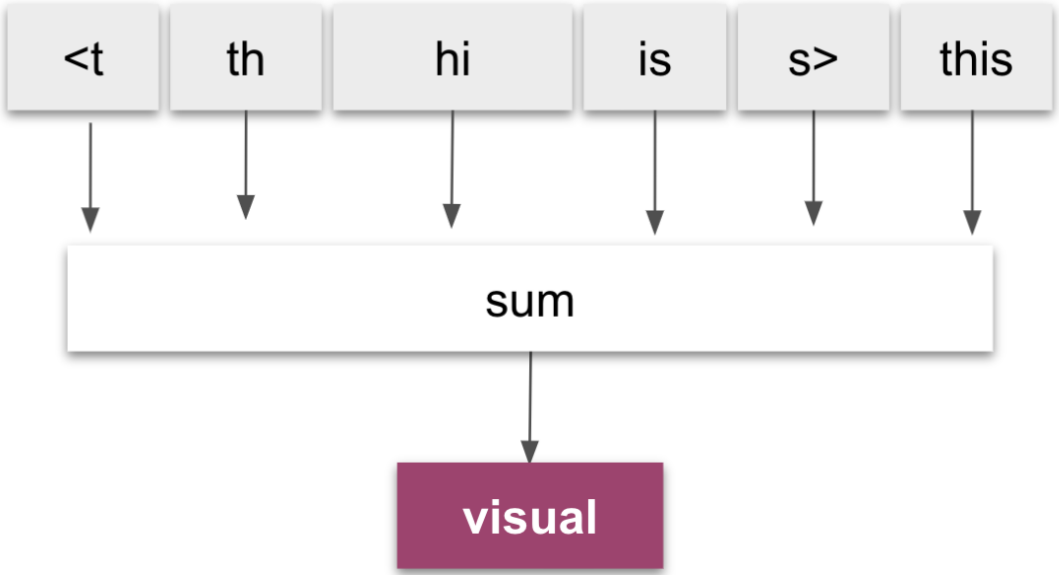
在参数 $\boldsymbol{\theta}$ 中，对于每一个词有两种类型的向量表示：

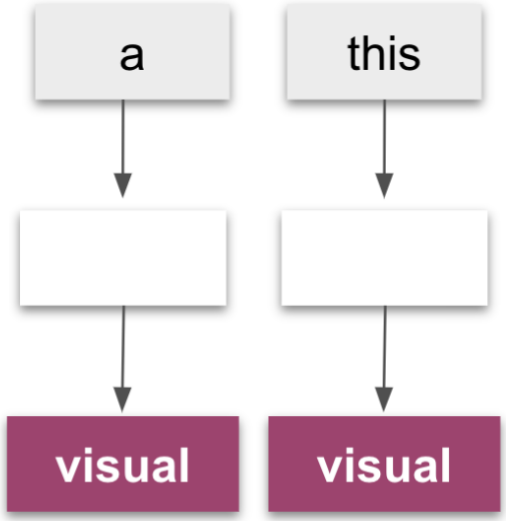
- v 表示输入向量，对于已知词，使用其输入向量；
- v' 表示输出向量，对待预测词，使用其输出向量。

在训练完成后，使用输入向量作为单词的表示。使用梯度下降或其他优化算法训练模型。实际中，还会使用负采样（Negative Sampling）或层次 Softmax（Hierarchical Softmax）等技术来加速训练过程。

fastText

fastText 由 Facebook 研究团队于 2016 年提出，使用了 Word2Vec 的一些概念，但进行了改进和扩展，以便更好地处理稀有词汇和高效处理大规模数据集。

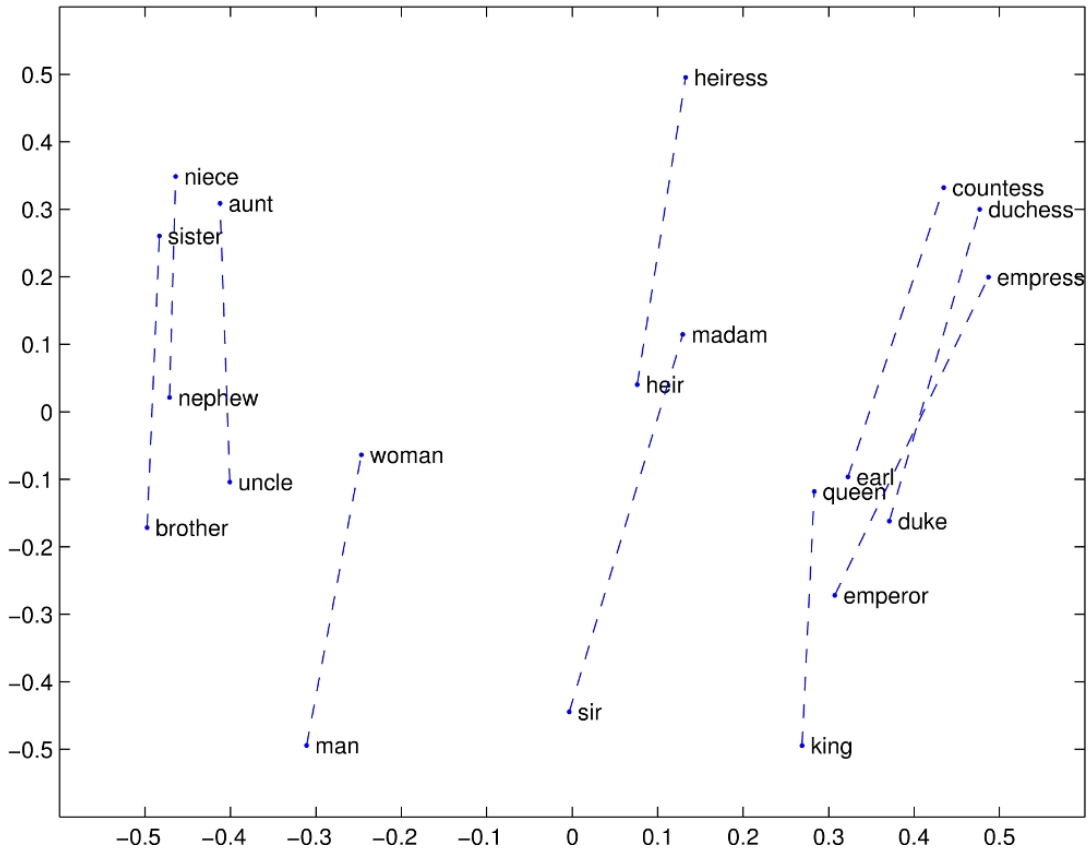




与 Word2Vec 模型直接学习单词向量不同，FastText 使用子词（n-grams）信息来表示单词，利用单词的内部结构来捕捉词义信息，更好地处理稀有词汇和拼写错误。

GloVe

GloVe 由斯坦福大学 NLP 研究团队于 2014 年提出，通过对词语共现矩阵进行分解，学习词汇表中每个单词的向量表示，优化目标为最小化预测共现概率与实际共现概率之间的平方误差，与 Word2Vec 相比，GloVe 更强调全局统计信息。下图中词对的矢量差 (性别) 大致相同。



Jaccard 相似度：基于集合的相似度度量，计算两个文本的词语集合 A 与 B 之间的交集与并集之比，适用于衡量文本的词汇重叠程度。

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

余弦相似度（Cosine similarity）：当文本表示为向量（例如词频、TF-IDF 或词嵌入）时，可以计算两个文本向量之间的余弦相似度。其关注向量的方向，对长度不敏感，因此适用于衡量文本的语义相似性。

$$\text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|_2 \cdot \|B\|_2} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}}$$

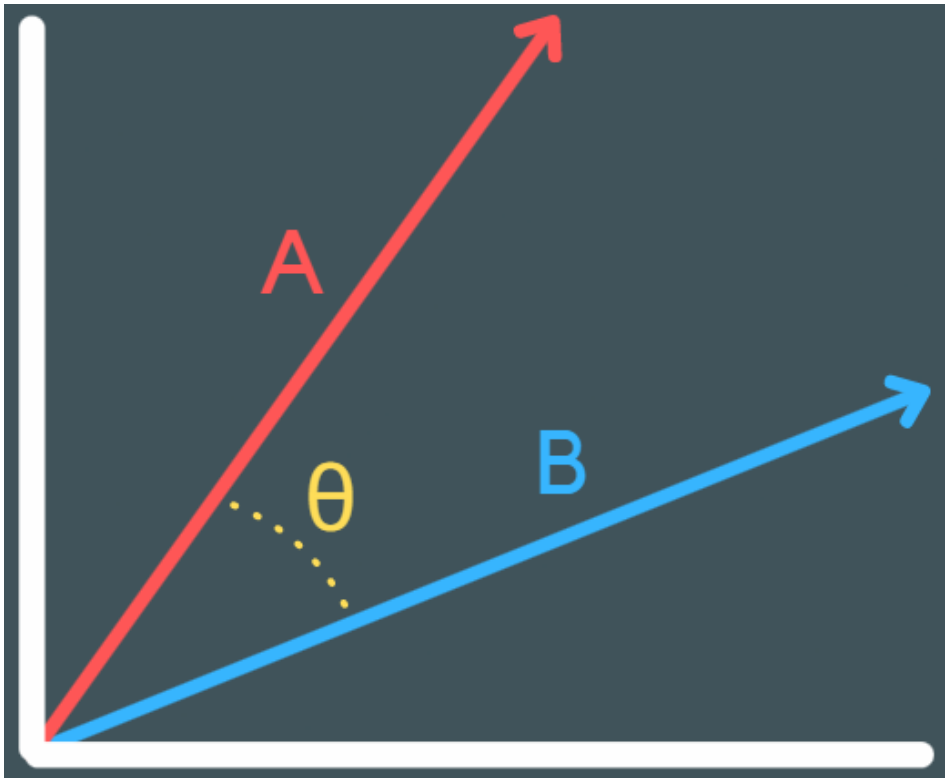
其中，A 和 B 为两个文本的向量。

欧氏距离：文本向量之间的欧氏距离越小，文本越相似。欧式距离对向量长度和方向均敏感，文本挖掘中，使用欧氏距离之前，通常将文本向量归一化。

$$\text{distance}(A,B) = \|A-B\|_2 = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\cdots+(a_n-b_n)^2}$$

归一化（L2 归一化）是将一个向量除以其 L2 范数（欧氏范数），使得归一化后的向量的 L2 范数为 1：

$$A' = \frac{A}{\|A\|_2} = \frac{A}{\sqrt{a_1^2+a_2^2+\cdots+a_n^2}}$$



余弦相似度与欧式距离之间的关系（文本向量归一化后）：

$$\text{distance}(A, B) = \sqrt{2} \times \sqrt{1 - \text{similarity}(A, B)}$$

即，单位圆上，角 θ 越大，其所对的弦越长。

相似度计算实例 I

文档 A: I like apples and oranges.
文档 B: I enjoy apples and bananas.
预处理: 去除标点，单词均转换为小写
词袋 A: {i: 1, like: 1, apples: 1, and: 1, oranges: 1, enjoy: 0, bananas:0}
词袋 B: {i: 1, like: 0, apples: 1, and: 1, oranges: 0, enjoy: 1, bananas:1}
词频向量 A: [1, 1, 1, 1, 1, 0, 0]
词频向量 B: [1, 0, 1, 1, 0, 1, 1]

1. Cosine Similarity:

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\|_2 \cdot \|B\|_2} = \frac{3}{\sqrt{5} \cdot \sqrt{5}} \approx 0.6$$

2. Euclidean Distance:

相似度计算实例 II

$$\text{distance}(A, B) = \|A - B\|_2 = \sqrt{6}$$

3. Jaccard Similarity:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{3}{7} \approx 0.43$$

总结

① 词语之间的相似度只能用词嵌入向量进行计算，因为独热词向量之间始终正交；

② 上述相似度均可用于计算文本之间的相似度，向量可采用 TF-IDF，词频向量，词嵌入向量 (通过求和或者平均文本中包含的词的向量)；

③ 由词袋表示的文本可以计算 Jaccard 相似度。

1. 使用 NLTK 语料库 API 和提供的历年政府工作报告文本，构建政府工作报告语料库。

2. 结合本讲文本统计量的内容，探索历年政府工作报告中用词特点，结果以图或表方式呈现，并据此给出你的分析。

THE END