第三章 词袋表示

1.1 向量空间模型和词袋表示（word、N-gram）

1.2 权重计算策略 （binary\frequency\TF-IDF）

1.3 具体实现：基于sklearn的向量化

文本表示

- 怎么表示一个文档?
  - 使其可以进行计算
  - 保留文档之间的关系，或者文档内部的结构?

The **University of Virginia** (**UVA** or **U.Va.**), often referred to as simply **Virginia**, is a public research university in Charlottesville, Virginia. UVA is known for its historic foundations, student-run honor code, and secret societies.

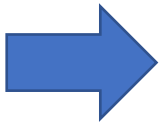
Its initial Board of Visitors included U.S. Presidents Thomas Jefferson, James Madison, and James Monroe. President Monroe was the sitting President of the United States at the time of the founding; Jefferson and Madison were the first two rectors. UVA was established in 1819, with its Academical Village and original courses of study conceived and designed entirely by Jefferson. UNESCO designated it a World Heritage Site in 1987, an honor shared with nearby Monticello.[4]

表示模型

原始文本数据不能直接输入到机器学习算法；

文本表示-把原始文本数据转换成算法能够处理的数值化的标准形式，转换过程中尽量保留原始数据的有用信息从而得到有意义的分析结果。

Shallow parsing, also known as light parsing or chunking.
It is a technique of analyzing the structure of a sentence to break it down into its smallest constituents。。。



```
([[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
  [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
  [0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
    [1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0],
   [1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
    [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1],
    [0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0]]
```

非结构化->结构化

向量空间模型（vector space model）

- 把一个文本文档（一则新闻、一个评论、一个网页）表示成向量，其中每个分量表示文档中的一个特征。假设一个包含n个文档的数据集总共涉及 p个特征，那么每个文档表示成p维空间的一个向量， 即

$$d_i = [w_{i1}, w_{i2}, \dots w_{ip}]$$

其中w_{ij}是第j个特征在第i个文档中的权重。

这些特征是什么？

词袋(Bag-of-Words)

- 把单个词作为特征构成向量空间
  - Doc1: Text mining is to identify useful information.
  - Doc2: Useful information is mined from text.
  - Doc3: Apple is delicious.

	text	information	identify	mining	mined	is	useful	to	from	apple	delicious
Doc1	1	1	1	1	0	1	1	1	0	0	0
Doc2	1	1	0	0	1	1	1	0	1	0	0
Doc3	0	0	0	0	0	1	0	0	0	1	1

- 每个词对应一个维度，即词典的大小决定了该特征空间的维数。
- 由于每个文档只包含词典中的少部分词，词袋模型得到的文本数据表示成高维稀疏数据。

向量空间模型表示：预处理+词袋

Doc1: '*Text mining is to identify useful information.*'

1. 分词Tokenization:

Doc1: *'Text', 'mining', 'is', 'to', 'identify', 'useful', 'information', '.'*

2. 词干提取Stemming:

Doc1: *'text', 'mine', 'is', 'to', 'identify', 'use', 'inform', '.'*

3. 去停用词Stopword:

Doc1: *'text', 'mine', 'identify', 'use', 'inform'*

Doc2: Useful information is mined from text. ➡ *'use', 'inform', 'mine', 'text'*

Doc3: Apple is delicious. ➡ *'apple', 'delici'*

	text	inform	identify	mine	use	apple	delici
Doc1	1	1	1	1	1	0	0
Doc2	1	1	0	1	1	0	0
Doc3	0	0	0	0	0	1	1

➡ Binary(二值或布尔型)，代表是否出现。

词袋模型中每个token的权重

■权重信息的重要性

- 从整个文档数据集来看：有些词包含更重要的文档内容信息
- 从每个文档来看：不是所有的词都一样重要

■如何衡量token权重?

- 两种基本方法：
  - 与token在某个文档出现的次数成正比
  - 与包含该token文档数目成反比

词频(Term frequency)

- 基本思想：如果一个词出现次数多说明它重要
- 基本TF计算：$tf(t,d)$ 等于该词 t 文档 d中出现的次数

  - 不同文档的长度不同， 重复出现的信息比第一次出现包含的信息少。

  对文档长度进行适当惩罚，得到规范化的词频（Normalized TF）。

IDF: Inverse document frequency

- 基本思想: 比起出现在很多个文档的那些词，一个只出现在少数文档的词更具有辨别力。
- 方法
  - **对稀有词赋给更高权重，公式：**

$$IDF(t) = 1 + \log\left(\frac{N}{df(t)}\right)$$

非线性缩放

数据集包含的文档总数

包含 t的文档的数目

  - 该项反映的是一个词在整个数据集层面的性质，与单个文档没有直接关系。
  - 在实现时，可以在分母加上1，防止除数为0。

TF-IDF

- 结合 TF 和 IDF
  - **在某个文档出现次数多 → 高 tf → 权重大**
  - **在整个数据集中稀有→ 高 idf→ 权重大**

$$w(t,d) = TF(t,d) \times IDF(t)$$

- 是信息检索中用得最多的权重计算方法! (G Salton et al. 1983)

有时候对得到的tf-idf向量进行 l_2 范数归一化，得到单位向量。

词袋表示-实现

	Document	Category
0	The sky is blue and beautiful.	weather
1	Love this blue and beautiful sky!	weather
2	The quick brown fox jumps over the lazy dog.	animals
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans	food
4	I love green eggs, ham, sausages and bacon!	food
5	The brown fox is quick and the blue dog is lazy!	animals
6	The sky is very blue and the sky is very beautiful today	weather
7	The dog is lazy but the brown fox is quick!	animals

样本语料（corpus）

基于sklearn实现向量化

```
from sklearn.feature_extraction.text import CountVectorizer
# 稀疏矩阵表示
cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(corpus)
# 查看矩阵的非零值
print(cv_matrix)
```

定义模型（选择算法、设置超参数）

```
  (0, 23)       1
  (0, 22)       1
  (0, 14)       1
  (0, 4)        1
  (0, 0)        1
  (0, 3)        1
  (1, 22)       1
  (1, 4)        1
  (1, 0)        1
```

结果以稀疏矩阵存储，如(0, 23)　1表示第23个词在第0个文档出现了1次。

权重为对应词在该文档出现的次数。

建立/训练模型（得到模型）

CountVectorizer:

把文本进行词袋表示，得到词频矩阵。
min_df、max_df用来过滤token: 一个token出现的文档数如果小于min_df，或大于max_df，则它就会被过滤。

binary=True，则把次数转成1，0布尔型，默认false。

其他参数：
stop_words- 去掉stopwords
tokenizer-指定（外部）分词算法

基于sklearn实现向量化

```
vocab = cv.get_feature_names()
cv_matrix = cv_matrix.toarray()
pd.DataFrame(cv_matrix, columns=vocab)
```

注意：此处直接对原始文档进行向量化表示，未进行任何预处理。真实应用中一般先预处理再向量化。

	and	bacon	beans	beautiful	blue	breakfast	brown	but	dog	eggs	...	love	over	quick	sausages	sky	the
0	1	0	0	1	1	0	0	0	0	0	...	0	0	0	0	1	1
1	1	0	0	1	1	0	0	0	0	0	...	1	0	0	0	1	0
2	0	0	0	0	0	0	1	0	1	0	...	0	1	1	0	0	2
3	1	1	1	0	0	1	0	0	0	1	...	0	0	0	1	0	0
4	1	1	0	0	0	0	0	0	0	1	...	1	0	0	1	0	0
5	1	0	0	0	1	0	1	0	1	0	...	0	0	1	0	0	2
6	1	0	0	1	1	0	0	0	0	0	...	0	0	0	0	2	2
7	0	0	0	0	0	0	1	1	1	0	...	0	0	1	0	0	2

8 rows × 28 columns

doc-term matrix： 每行对应一个文档，每列对应一个单词

基于N-grams的词袋模型

- 词袋：假设所有token之间相互独立，未能考虑顺序信息

改进方法：以N-grams为单位构建词袋表示
N-grams: 由N 个tokens组成的连续序列

如‘*Text mining is to identify useful information.*’的Bigrams为：
‘*text_mining*’, ‘*mining_is*’, ‘*is_to*’, ‘*to_identify*’, ‘*identify_useful*’, ‘*useful_information*’, ‘*information_.*’

好处: 可以描述token之间的局部相关性或顺序
不足: 纯粹从统计角度来考虑，且增加词汇量 $O(V^N)$

基于sklearn实现n-gram向量化

```
# 把 ngram_range 设置成 (2,2) 得到bigrams
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(corpus)
bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

	and bacon	and beans	and beautiful	and the	bacon eggs	beautiful sky	beautiful today	blue and	blue dog
0	0	0	1	0	0	0	0	1	0
1	0	0	1	0	0	1	0	1	0
2	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	1

把 ngram_range 设置成 (1,n) 同时得到unigram、bigrams…ngrams，即词典中混合了多种长度的token。
当n比较大，出现的次数会很少，所以n一般不超过3。

TfidfTransformer：从词频矩阵计算TF-IDF矩阵

```
from sklearn.feature_extraction.text import TfidfTransformer
#norm='12'表示归一化到单位向量
tt = TfidfTransformer(norm='12', use_idf=True)
tt_matrix = tt.fit_transform(cv_matrix)
tt_matrix = tt_matrix.toarray()
vocab = cv.get_feature_names()
pd.DataFrame(np.round(tt_matrix, 2), columns=vocab)
```

	and	bacon	beans	beautiful	blue	breakfast	brown	but	dog	eggs	...	love	over	quick
0	0.32	0.00	0.00	0.47	0.41	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.00
1	0.27	0.00	0.00	0.39	0.34	0.00	0.00	0.00	0.00	0.00	...	0.45	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.30	0.00	0.30	0.00	...	0.00	0.41	0.30

直接从文本得到TF-IDF向量

TfidfVectorizer：把文本转成tfidf矩阵。

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(min_df=0., max_df=1., norm='l2', use_idf=True, smooth_idf=True)
tv_matrix = tv.fit_transform(corpus)
tv_matrix = tv_matrix.toarray()
vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

$$IDF(t) = 1 + \log(\frac{N}{\mathbf{1}+df(t)})$$

smooth_idf=True表示计算idf时分母加上1，避免分母为0。

对新的文档进行向量化

在实际应用中，模型训练好以后，在测试或使用阶段需要对新的文档进行向量化。

```
new_doc = 'the sky is green today'
pd.DataFrame(np.round(tv.transform([new_doc]).toarray(), 2),columns=tv.get_feature_names())
```

]:

	and	bacon	beans	beautiful	blue	breakfast	brown	but	dog	eggs	...	love	over	quick	sausages	sky	the
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.4	0.31

1 rows × 28 columns

Sklearn里面：

- 训练模型：fit_transform(train_data)
- 把已有模型用到新的数据：transform(test_data)

对中文的向量化表示

```
import jieba
from sklearn.feature_extraction.text import  TfidfVectorizer
def jieba_tokenize(text):
    return jieba.cut(text)
tfidf_vectorizer = TfidfVectorizer(tokenizer=jieba_tokenize,lowercase=False)
text_list = ["为了计算机更好理解数据",
"大规模数字化（中文）信息资源组织的基本流程也如下",
"特征工程的意义：直接影响预测结果",
"数据的特征抽取"]
tt_matrix = tfidf_vectorizer.fit_transform(text_list)
tt_matrix=tt_matrix.toarray()
pd.DataFrame(np.round(tt_matrix,2))
```

词袋模型的优缺点

- 简单、经常在信息检索中广泛应用
- 纯粹基于统计信息，忽略了词之间的顺序、语义等重要信息。