

人工智能及其实践教程

自然语言处理

1 什么是自然语言处理

2 文本分词

3 使用stemming还原词汇

4 基于词义的词形还原

5 文本分块

6 使用词袋模型提取词频矩阵

7 案例:构建一个性别识别器

8 总结

➤ 什么是自然语言处理

➤ 文本分词

➤ 使用stemming还原词汇

➤ 使用lemmatization还原词汇

➤ 文本分块

➤ 使用词袋模型提取文章中的词频
矩阵

➤ 构建性别识别器

1 什么是自然语言处理

自然语言处理（**Natural Language Processing**）是计算机科学领域与人工智能领域中的一个重要方向。它研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法。自然语言处理是一门融语言学、计算机科学、数学于一体的科学。

自然语言处理(NLP)已经成为现代系统的重要组成部分。它广泛应用于搜索引擎、人机对话接口、文档处理等。机器可以很好地处理结构化数据。但是，当涉及到无固定形式的文本时，它将很难处理。**NLP**的目标是开发一种算法，使计算机能够理解无结构的文本，并帮助他们理解这种语言。

1 什么是自然语言处理



自然语言处理的应用

1 什么是自然语言处理

本章学习需要提前做的是：

- 导入Python的第三方库NLTK(Natural Language Toolkit)
- 下载NLTK提供的数据集

在终端下进入Python环境，并输入以下代码来下载这些数据集：

```
>>> import nltk
```

```
>>> nltk.download()
```

- 安装gensim包

2 文本分词

当我们处理文本时，我们需要将它拆分成小片来进行分析。它是将输入的文本分成像单词或句子的一小片，这些片被称之为**原型**。我们可以根据自己所想，定义自己的方法将文本分成许多片。

下面我们用一段程序来实现文本的分词：

2 文本分词

程序 15.1 文本分词

```
1: from nlk.tokenize import sent_tokenize, word_tokenize, WordPunctTokenizer
2:
3: input_text = "Do you know what natural language processing is? This is a very
4: interesting technology! We'll look at it in this section."
5:
6: print("\nSentence tokenizer:")
7: print(sent_tokenize(input_text))
8:
9: print("\nWord tokenizer:")
10: print(word_tokenize(input_text))
11:
12: print("\nWord punct tokenizer:")
13: print(WordPunctTokenizer().tokenize(input_text))
```

导入一个名为NLTK的程序包，用于导入不同形式的分词器。

定义了将用于分词的输入文本。

之后我们分别使用不同的分词器对输入文本进行分词。

2 文本分词

输出:

Sentence tokenizer:

```
['Do you know what natural language processing is?', 'This is a very interesting technology!',  
"We'll look at it in this section."]
```

Word tokenizer:

```
['Do', 'you', 'know', 'what', 'natural', 'language', 'processing', 'is', '?', 'This', 'is', 'a', 'very', 'interesting  
'technology', '!', 'We', "I'll", 'look', 'at', 'it', 'in', 'this', 'section', '.']
```

Word punct tokenizer:

```
['Do', 'you', 'know', 'what', 'natural', 'language', 'processing', 'is', '?', 'This', 'is', 'a', 'very', 'interesting  
'technology', '!', 'We', '', 'I'll', 'look', 'at', 'it', 'in', 'this', 'section', '.']
```

3 使用stemming还原词汇

对于一些变化的词汇，我们必须处理相同单词的不同形式，并且使计算机能够明白这些不同的单词有相对的形式。例如，单词**write**能够以很多形式出现，如下图所示：

当我们分析文本时，提取这些基本形式是很有用的。它将使我们能够提取有用的统计信息来分析输入文本。词干提取(**stemming**)能够做到这一点。词干提取器(**stemmer**)的目的是通过将单词的不同形式转换为基本形式来减少单词量。去掉单词的尾部将其变成基本形式是一个启发式的过程。

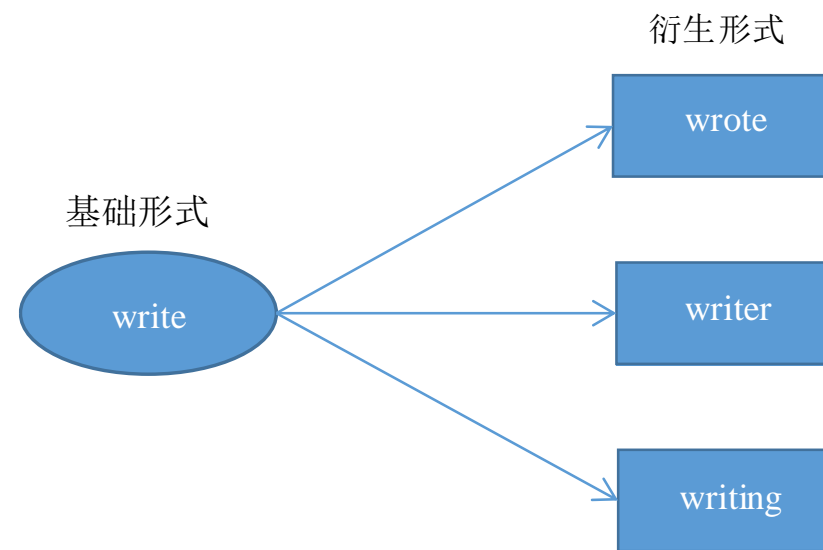


图15.3 write的衍生形式

3 使用stemming还原词汇


从程序来更加详细地了解 stemming 的原理。我们用'reading', 'calves', 'acting', 'undefined', 'house', 'possibly', 'version', 'hotel', 'learned', 'experience'这些单词作为测试，来看看stemming是如何工作的：

程序 15.2 使用 stemming 还原词汇



```
1:  from nltk.stem.porter import PorterStemmer
2:  from nltk.stem.lancaster import LancasterStemmer
3:  from nltk.stem.snowball import SnowballStemmer
4:
5:  input_words = ['reading', 'calves', 'acting', 'undefined', 'house',
6:                'possibly', 'version', 'hotel', 'learned', 'experience']
7:
8:  porter = PorterStemmer()
9:  lancaster = LancasterStemmer()
10: snowball = SnowballStemmer('english')
11:
12: stemmer_names = ['Porter', 'Lancaster', 'Snowball']
13: formatted_text = '{:>16}' * (len(stemmer_names) + 1)
14: print('\n', formatted_text.format('Input Word', *stemmer_names),
15:       '\n', '**70)
16:
17: for word in input_words:
18:     output = [word, porter.stem(word),
19:               lancaster.stem(word), snowball.stem(word)]
20:     print(formatted_text.format(*output))
```

3 使用stemming还原词汇

导入3个不同的词干提取器：
Porter、Lancaster、Snowball



创建三个提取器对象
创建了一个显示表格并格式化
输出文本



程序 15.2 使用 stemming 还原词汇

```
1: from nltk.stem.porter import PorterStemmer
2: from nltk.stem.lancaster import LancasterStemmer
3: from nltk.stem.snowball import SnowballStemmer
4:
5: input_words = ['reading', 'calves', 'acting', 'undefined', 'house',
6:                'possibly', 'version', 'hotel', 'learned', 'experience']
7:
8: porter = PorterStemmer()
9: lancaster = LancasterStemmer()
10: snowball = SnowballStemmer('english')
11:
12: stemmer_names = ['Porter', 'Lancaster', 'Snowball']
13: formatted_text = '{:>16}' * (len(stemmer_names) + 1)
14: print('\n', formatted_text.format('Input Word', *stemmer_names),
15:       '\n', '**70)
16:
17: for word in input_words:
18:     output = [word, porter.stem(word),
19:              lancaster.stem(word), snowball.stem(word)]
20:     print(formatted_text.format(*output))
```

4 基于词义的词形还原

Lemmatization是另一种词形还原的方式。在前一节中，我们看到从词干中提取词的基本形式有时候可能没有任何意义。**Lemmatization**采取了一种更具结构化的方法解决了这个问题。

Lemmatization原理是使用语法和词态分析器进行单词分析。它包含去除了如ing和ed等后缀的单词的基本形式。所有基本形式的单词集合被称作字典。

这里，我们将使用**Lemmatization**来对上一节例子中的单词进行词形还原，进而比较它们之间的区别。我们分别使用名词还原器和动词还原器对单词进行词形还原。请看下面程序：

4 基于词义的词形还原

程序 15.3 使用 lemmatization 还原词汇

```
1: from nltk.stem import WordNetLemmatizer
2:
3: input_words = ['reading', 'calves', 'acting', 'undefined', 'house',
4:                'possibly', 'version', 'hotel', 'learned', 'experience']
5:
6: lemmatizer = WordNetLemmatizer()
7:
8: lemmatizer_names = ['Noun Lemmatizer', 'Verb Lemmatizer']
9: formatted_text = '{:>24}' * (len(lemmatizer_names) + 1)
10: print('\n', formatted_text.format('Input Word', *lemmatizer_names),
11:       '\n', '*'*74)
12:
13: for word in input_words:
14:     output = [word, lemmatizer.lemmatize(word, pos='n'),
15:              lemmatizer.lemmatize(word, pos='v')]
16:     print(formatted_text.format(*output))
```

8-11行是创建显示列表并格式化文本。

13-16行遍历输入单词并分别使用动词还原器和名词还原器来还原词汇。

4 基于词义的词形还原

输出:		
Input Word	Noun Lemmatizer	Verb Lemmatizer

reading	reading	read
calves	calf	calve
acting	acting	act
undefined	undefined	undefined
house	house	house
possibly	possibly	possibly
version	version	version
hotel	hotel	hotel
learned	learned	learn
experience	experience	experience

可以看到，当遇到形如reading或者calves这些单词时，名词还原器和动词还原器的还原结果是不一样的。如果将这些输出与之前的词干提取器的输出结果相比，这两者的结果也有不同。Lemmatizer输出都是有意义的，而stemmer输出可能有意义也可能没有意义。

5 文本分块

文本数据经常需要被分成一小块来进行分析，这个过程称之为**分块**。这种技术在文本分析中使用频繁。使用文本分块的情况变化很多，各不相同，这依赖于手头的项目。文本分块与分词不同。在分块时，我们不受任何条件的限制，并且输出的结果是有意义的。

下面我们来实现一段程序，用于将输入的文本进行分块：

程序 15.4 文本分块

```
1: import numpy as np
2: from nltk.corpus import brown
3:
4: def chunker(input_data, N):
5:     input_words = input_data.split(' ')
6:     output = []
7:     cur_chunk = []
8:     count = 0
9:     for word in input_words:
10:         cur_chunk.append(word)
11:         count += 1
12:         if count == N:
13:             output.append(' '.join(cur_chunk))
14:             count, cur_chunk = 0, []
15:     output.append(' '.join(cur_chunk))
16:     return output
17:
18: if __name__ == '__main__':
19:     input_data = ' '.join(brown.words()[:6300])
20:     chunk_size = 800
21:     chunks = chunker(input_data, chunk_size)
22:     print('\nNumber of text chunks =', len(chunks), '\n')
23:     for i, chunk in enumerate(chunks):
24:         print('Chunk', i + 1, '==>', chunk[:50])
```

5 文本分块

输出：

Number of text chunks = 8

Chunk 1 ==> The Fulton County Grand Jury said Friday an invest

Chunk 2 ==> 2 , 1913 . They have a son , William Berry Jr. , a

Chunk 3 ==> bonds . Schley County Rep. B. D. Pelham will offer

Chunk 4 ==> Texas was a republic " . It permits the state to

Chunk 5 ==> the requirement that each return be notarized . In

Chunk 6 ==> the Education courses . Fifty-three of the 150 rep

Chunk 7 ==> drafts of portions of the address with the help of

Chunk 8 ==> plan alone would boost the base to \$5,000 a year a

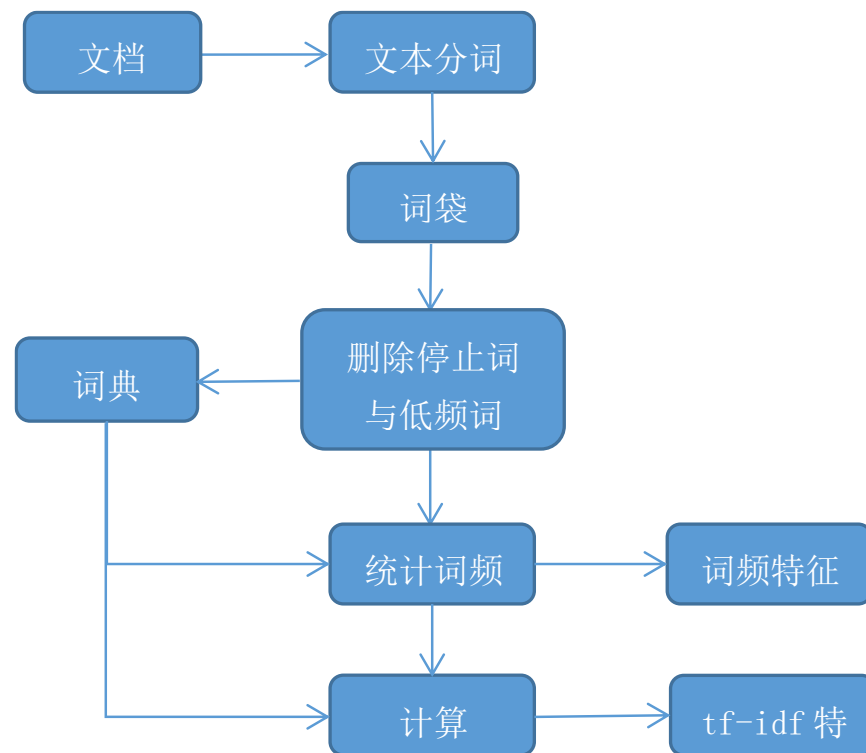
6 使用词袋模型提取词频矩阵

Bag of Words，也称作“词袋”。它用于描述文本的一个简单数学模型，也是常用的一种文本特征提取方式。我们编写一段程序来使用词袋模型提取一个词频矩阵：

程序 15.5 使用词袋模型提取词频矩阵

```
1: import numpy as np
2: from sklearn.feature_extraction.text import CountVectorizer
3: from nltk.corpus import brown
4: from text_chunker import chunker
5:
6: input_data = ''.join(brown.words()[:5600])
7: chunk_size = 900
8: text_chunks = chunker(input_data, chunk_size)
9:
10: chunks = []
11: for count, chunk in enumerate(text_chunks):
12:     d = {'index': count, 'text': chunk}
13:     chunks.append(d)
```

词袋模型应用的基本流程



从Brown语料库中读入5600个单词，并定义每块的单词数为900，使用chunker函数将文本进行分块。

10-13行将所分的块转换为字典项。

6 使用词袋模型提取词频矩阵

```
15: count_vectorizer = CountVectorizer(min_df=7, max_df=18)
16: document_term_matrix = count_vectorizer.fit_transform([chunk['text'] for
17:                                                         chunk in chunks])
18:
19: vocabulary = np.array(count_vectorizer.get_feature_names())
20:
21: chunk_names = []
22: for i in range(len(text_chunks)):
23:     chunk_names.append('Chunk ' + str(i + 1))
24:
25: print("\nDocument Term Matrix:")
26: formatted_text = '{:>9}' * (len(chunk_names) + 1)
27: print('\n', formatted_text.format('Word', *chunk_names), '\n', '*'*76)
28: for word, item in zip(vocabulary, document_term_matrix.T):
29:     output = [word] + [str(freq) for freq in item.data]
30:     print(formatted_text.format(*output))
```

使用CountVectorizer方法对单词进行计数，并提取单词矩阵。

该方法需要两个输入参数，第一个参数是出现在文档中单词的最小频率度，第二个参数是出现在文档中的单词的最大频率度。

6 使用词袋模型提取词频矩阵

结果显示:

输出:							
Document Term Matrix:							
Word	Chunk 1	Chunk 2	Chunk 3	Chunk 4	Chunk 5	Chunk 6	Chunk 7

*							
and	27	5	15	8	16	17	6
are	2	2	1	1	4	1	1
as	6	4	4	2	9	3	3
be	6	11	5	10	2	3	2
by	3	5	4	10	10	7	3
for	9	12	5	13	7	5	2
his	4	5	5	2	2	5	1
in	15	16	12	14	19	20	3
is	3	7	5	1	7	5	1
it	9	6	12	6	1	3	2
of	33	24	28	36	37	33	3
on	4	6	3	15	1	6	4
one	1	3	1	4	1	1	1
or	2	2	1	1	2	1	1
the	77	57	48	58	48	72	9
to	11	34	22	26	18	19	5
was	5	8	9	5	6	6	1
with	2	2	4	1	3	3	1

7 案例:构建一个性别识别器

性别识别是一个有趣的问题。既然如此，我们将使用启发式的方法来构建一个特征向量，并且使用它训练一个分类器。这里使用的启发式是被给定名字的最后N个字母。

假设我们要对Sophia, Edward, William, Shirley这四个名字进行分析，进而判断他们的性别。我们用资料库中已有的训练数据来创建一个朴素贝叶斯分类器，再使用这个分类器对这些名字进行测试。我们把N的值分别设置为1、2、3、4、5。使用NLTK(自然语言处理工具包)中提供的的内置方法来计算分类器的准确性。看看程序是如何执行的：

程序 15.5 使用词袋模型提取词频矩阵

```
1: import random
2: from nltk import NaiveBayesClassifier
3: from nltk.classify import accuracy as nltk_accuracy
4: from nltk.corpus import names
5:
6: def extract_features(word, N=2):
7:     last_n_letters = word[-N:]
8:     return {'feature': last_n_letters.lower()}
```

首先，我们导入了random模块，再从nltk库中导入朴素贝叶斯分类器和测试分类器准确度的模块，最后导入语料库中的names模块。

定义一个函数，来从输入的词汇中提取最后N个字母。

8 总结

在这一章中，我们学习了关于各种**自然语言处理**的基本概念。我们讨论了分词以及如何将输入文档分离成多个词。我们学习了如何使用**stemming**和**lemmatization**将单词还原成基本形式。

我们讨论了什么是词袋模型，并且为输入的文本构建了一个文档的单词矩阵，我们之后学习了怎样使用机器学习进行文本的分类。我们还使用启发式构建了一个性别识别器，使用机器学习分析影评。最后，我们讨论了基于**LDA**的主题建模。