上节课回顾

回顾:

- 文本分析与挖掘与数据挖掘、自然语言处理的关系是什么?
- 文本作为处理对象，有什么难的地方?
- 文本分析与挖掘的应用-给出几个例子?
- 文本挖掘的基本步骤?
- 为什么要选这门课？你希望在这门课上学到什么?
- 你记得的文本预处理有哪些？中英文一样吗?

第二章 文本预处理

1.1 分词
1.2 词干提取
1.3 词性还原
1.4 去停用词
1.5 词性标注

本章重点:
- 理解文本预处理的重要性以及常用预处理;
- 掌握基于nltk实现常用英文预处理,以及jieba中文分词。

为什么要预处理？

- 原始文本是一种包含噪声的（如HTML标签）、非结构化的数据；
- 需要通过清洗、规范化以及专门的文本预处理步骤来增加内容有效性。

“Garbage in Garbage out!”
没有效的预处理可能无法得到有意义的分析结果。

文本预处理

英文文本：

- 分词(Tokenization)：把句子分成有意义的单位(token)。
- 词干提取(Stemming)：抽取词的词干或词根形式（不一定能够表达完整语义），如“revival”词干提取的结果为“reviv”
- 词性还原(Lemmatization)：把一个任何形式的语言词汇还原为一般形式（能表达完整语义），如”spoken”还原为”speak”
- 去停用词(Stopword removal)：把句子中的”is”，“of”，“a”等常见的不具有特殊意义的词去掉。
- 词性标注(POS tagging)：对分词结果进行名词、动词、形容词、代词等标注。

中文文本预处理的关键是分词！

分词（tokenization/segmentation）

- 词语通常作为基本的文本单位，构成短语->从句->句子->段落。
- 分词：把一个句子分割成一个以词（token）为单位的序列。
- 分词结果直接影响到后面的处理和表示；

有分隔语言vs无分割语言

- 词之间包含分割符（空格）的语言，如英语、德语、法语等
- 不具有词分割符的语言，如汉语、日语

拓展：在英文中，单词(word)是最小的具有独立性的语言单位，语素(morpheme)是最小语言单位。近几年开始基于subwords为单位进行分词。比如Byte Pair Encoding (BPE)，起初用在机器翻译。在Google 开发的sentence piece中有该算法实现，用在BERT等预训练语言模型。

英文分词（基于nltk）

In [9]:
```
import nltk
import numpy as np
default_wt=nltk.word_tokenize
sample_text="""word tokenization is the process of splitting or
 segmenting sentences into their constituent words."""
words=default_wt(sample_text)
np.array(words)
```

也可以用nltk.WordPunctTokenizer()

Out[9]:
```
array(['word', 'tokenization', 'is', 'the', 'process', 'of', 'splitting',
       'or', 'segmenting', 'sentences', 'into', 'their', 'constituent',
       'words', '.'], dtype='<U12')
```

除了默认的word_tokenize，以及WordPunctTokenizer，还可以用ToktokTokenizer

from nltk.tokenize.toktok import ToktokTokenizer

中文分词（基于jieba）

jieba具有多个模式。

```
import jieba
sample_text = "为什么文本分析和挖掘很有用呢？"
seg_list = jieba.cut(sample_text, cut_all=False)
print(u"[精确/默认模式]: ", "/ ".join(seg_list))
```

[精确/默认模式]: 为什么/ 文本/ 分析/ 和/ 挖掘/ 很/ 有用/ 呢/ ？

设置 cut_all=True 得到全模式结果

[全模式]: 为什么/ 什么/ 文本/ 本分/ 分析/ 和/ 挖掘/ 很/ 有用/ 呢/ ？

先分句后分词

以下函数利用NLTK同时实现句子和词两个单位的分割。

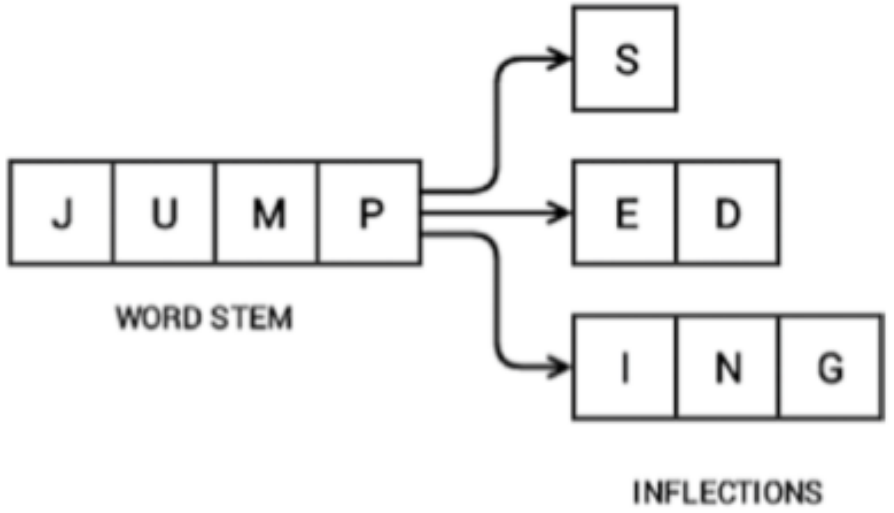
其他分句方法如PunktSentenceTokenizer

```
def tokenize_text(text):
      #对输入文本用nltk.sent_tokenize进行分句
    sentences=nltk.sent_tokenize(text)
      #对每个句子进行分词
    words=[nltk.word_tokenize(sentence) for sentence in sentences]
    return words
#调用以上函数
sample_text="""We also can leverage spaCy to perform tokenizations.
This should be enough to give you started with text tokenization."""
sents= tokenize_text(sample_text)
np.array(sents)
```

```
3]: array([list(['We', 'also', 'can', 'leverage', 'spaCy', 'to', 'perform', 'tokenizations', '.']),
       list(['This', 'should', 'be', 'enough', 'to', 'give', 'you', 'started', 'with', 'text', 'tokenization
', '.'])],
      dtype=object)
```

词干提取（stemming）

- 语素（mortheme）是自然语言中的最小独立单位，是构成词的要素，包含词干(stem)和词缀（前缀、后缀等）。
- 词干是构成一个词的基础，通过附加不同的词缀形成不同的词。
- 词干提取：提取出一个词中的词干。



词干提取

nltk实现了多个词干提取方法，包含在stem 模块中，Poerter Stemmer是比较流行的一个。

In [19]:
```
# Porter Stemmer
from nltk.stem import PorterStemmer
ps = PorterStemmer()

ps.stem('jumping'), ps.stem('jumps'), ps.stem('jumped')
```

Out[19]: ('jump', 'jump', 'jump')

In [20]:
```
ps.stem('strange')
```

Out[20]: 'strang'

注意：词干不一定是一个完整意义的词

词性还原(lemmatization)

In [21]:
```
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()
# 名词还原
print(wnl.lemmatize('cars', 'n'))
print(wnl.lemmatize('men', 'n'))
```

```
car
men
```

In [22]:
```
# 动词还原
print(wnl.lemmatize('running', 'v'))
print(wnl.lemmatize('ate', 'v'))
```

```
run
eat
```

In [23]:
```
# 形容词还原
print(wnl.lemmatize('saddest', 'a'))
print(wnl.lemmatize('fancier', 'a'))
```

```
sad
fancy
```

词性还原得到词的原型，是一个有意义的词。

注意:
左边例子中的"men" 并没有还原到"man"。

另外，nltk的WordNetLemmatizer需要用到词的POS词性标注。当词性不对时，结果不准确。

去停用词（stop-word removing）

文本中的停用词(stopword)如英文里面”a”，“the”，“and”，中文里面的“的”，“地”、“得”等不具有重要信息，一般在预处理时过滤掉从而减少词汇数。

不存在通用、完整的停用词表列，一般根据语言和具体应用场景来设置符合特定需求的列表。

比如在nltk默认的英语停用词表中，包括”not”,”no”这样的否定词，但这些词在情感分析中具有明确的情感成分，往往会被保留。所以确保不要把对目标任务有用的词过滤掉。

```
#查看nltk包含的默认停用词列表
nltk.corpus.stopwords.words('english')
```

用nltk得到不包含停用词的分词结果

```
from nltk.tokenize.toktok import ToktokTokenizer
tokenizer = ToktokTokenizer()
#nltk默认英语停用词表
stopword_list =nltk.corpus.stopwords.words('english')
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    #.strip()方法去掉字符串头尾多余的空格、\t, \n等字符。
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_lis
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
```

```
#从停用词列表中去掉"no"
stopword_list.remove('no')

#额外增加停用词
stopword_list =
stopword_list + ['one', 'two']
```

停用词表为小写，只会删除全部字母为小写的token。

In [45]:

```
#测试以上函数
remove_stopwords("The, and, if are stopwords, computer is not")
```

Out[45]: ', , stopwords , computer'

词性标注

POS(parts of speech)根据句法和词在句子中的作用对词进行类别标注。

词类别标注反映了多个词在构成一个句子时相互之间的关系以及各自在这个句子中的角色，对基于自然语言的应用提供了重要信息。

NLTK 和大多数其他标注模型都采用用了Penn Treebank 词类别来得到POS标注。

以下文档包含Penn Treebank对每个词类别的解释和例子

http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/Penn-Treebank-Tagset.pdf

词类别标注

基于nltk的 POS标注

```
import nltk
import pandas as pd
sentence = "US judge suspends Trump ban on TikTok downloads"
nltk_pos_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
pd.DataFrame(nltk_pos_tagged, columns=['Word', 'POS tag']).T
```

:

	0	1	2	3	4	5	6	7
Word	US	judge	suspends	Trump	ban	on	TikTok	downloads
POS tag	NNP	NN	VBZ	NNP	NN	IN	NNP	NNS

训练自己的词类别标注模型

- 把词类别标注看成一个分类任务，用标注好的数据作为训练集，基于ClassifierBasedPOSTagge类来训练一个自动标注模型。
- 利用nltk中的treebank中的标注好的语料进行训练和测试

```
#先导入数据，并分割成训练集和测试集
from nltk.corpus import treebank
data = treebank.tagged_sents()
train_data = data[:3500]
test_data = data[3500:]
print(train_data[0])
```

```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'),
('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), ('the', 'DT'), ('board',
'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'), ('No
v.', 'NNP'), ('29', 'CD'), ('.', '.')]
```

训练自己的词类别标注模型-默认方法

先看看nltk中的默认标注方法的效果。该方法把所有词都标注成指定的类型。

```
# 默认标注器
from nltk.tag import DefaultTagger
dt = DefaultTagger('NN')
# 测试集上的准确率
dt.evaluate(test_data)
```

```
0.1454158195372253
```

```
#对前面新闻题目的标注结果
dt.tag(nltk.word_tokenize(sentence))
```

```
[('US', 'NN'),
 ('judge', 'NN'),
 ('suspends', 'NN'),
 ('Trump', 'NN'),
 ('ban', 'NN'),
 ('on', 'NN'),
 ('TikTok', 'NN'),
 ('downloads', 'NN')]
```

训练自己的词类别标注模型-朴素贝叶斯

```
from nltk.classify import NaiveBayesClassifier
from nltk.tag.sequential import ClassifierBasedPOSTagger
nbt = ClassifierBasedPOSTagger(train=train_data,
                               classifier_builder=NaiveBayesClassifier.train)
# 测试集准确率、新闻题目例句的标注结果
print(nbt.evaluate(test_data))
print(nbt.tag(nltk.word_tokenize(sentence)))
```

```
0.9306806079969019
[('US', 'PRP'), ('judge', 'NN'), ('suspends', 'VBZ'), ('Trump', 'NNP'), ('ban', 'NN
'), ('on', 'IN'), ('TikTok', 'NNP'), ('downloads', 'NNS')]
```

预处理的局限性

- 如何选择恰当的预处理取决于语言种类、语料库, 以及应用场景。

分词(哪一种更好？):

- **Input:** It’s not straight-forward to perform so-called “tokenization.”
- **Output(1):** 'It’s', 'not', 'straight-forward', 'to', 'perform', 'so-called', '“tokenization.”'
- **Output(2):** 'It', '’', 's', 'not', 'straight', '-', 'forward, 'to', 'perform', 'so', '-', 'called', ‘“', 'tokenization', '.', '”‘

- 进行词干提取/词性还原有助于建立词之间的相关性，但丢失了部分更加准确的信息。
- 去停止词可以减少词汇量，但也可能破坏句子原有意思和结构。

如 this is not a good option -> option

课后作业

- 下载和安装nltk（如果是anaconda默认已经安装）、spacy、以及 jieba；用import 对应的库来测试是否安装成功，如import nltk。
- 另外需要下载nltk data、spacy的英语包（en_core）

  import nltk

  nltk.download()　(nltk对应语料http://www.nltk.org/data.html)
- spacy　（https://spacy.io/）

  pip install spacy

  nlp = spacy.load(“en_core_web_sm”)　(spacy对应的语料)
- jieba

  pip install jieba

在线导入nltk_data和en_core有问题的参考后面的线下手动配置

其他相关的包（在线安装）

打开anaconda命令行输入（anaconda prompt）

- nltk(anacond默认已经安装)

  import nltk （import 不报错说名nltk已经安装)

  nltk.download() （nltk对应语料http://www.nltk.org/data.html)

- spacy （https://spacy.io/）

  pip install spacy

  nlp = spacy.load(“en_core_web_sm”) （spacy对应的语料)

- jieba

  pip install jieba

- gensim

  pip install gensim

用import 来测试是否已经安装，如import nltk

线下导入和配置

- 在线安装nltk_data和en_core有问题的，从钉钉群文件夹中下载放到Anaconda3目录下。
- 把nltk_data压缩文件全部解压，包括nltk_data目录里面的压缩包。
- 对en_core直接在命令行用以下命令安装(安装成功后jupyter可能要重启):

```
pip install D:\Anaconda3\en_core_web_sm-2.2.5.tar.gz
```

预告:
10月22日将进行第一次上机实验。内容为文本预处理。