



华中师范大学

自然语言处理

第四章 中文分词

主讲教师：曾江峰

华中师范大学 信息管理学院

jfzeng@ccnu.edu.cn

中文分词

- ❖ 分词是指将连续的字序列按照一定的规范重新组合成词序列的过程。在英文中，单词之间是以空格作为自然分界符，而中文词与词之间没有明显的界限标志，只有明显的逗号、句号等分界符进行句段划界，因此分词是汉语文本分析处理中的首要问题，是机器翻译、语音合成、自动分类、自动摘要、自动校对等中文信息处理的基础。
- ❖ 本章介绍了中文分词的特点，基于规则和词表以及基于统计等中文分词方法，重点讲解了jieba分词库和HanLP分词库。



中文分词

- 1 概述
- 2 常见中文分词方法
- 3 中文分词困惑
- 4 jieba分词库
- 5 HanLP分词

1. 概述

特点：

- 分词规范
- 歧义词切分
- 未登录词识别



1. 概述

特点：

- 分词规范

中文因其自身语言特性的局限，字（词）的界限往往很模糊，关于字（词）边界的划定尚没有一个公认的、权威的标准，导致汉语分词难度极大。



1. 概述

特点:

- 歧义词切分

- (1) **交集型切分歧义**，AJB类型满足AJ和JB分别成词。例如，“大学生”一词的一种切分方式“大学/生”，另一种切分方式“大/学生”
- (2) **组合型切分歧义**，AB满足A和B、AB分别成词。例如，“才能”的一种切分为“/高超/的/才能”，另一种切分“坚持/才/能/成功”。
- (3) **混合型切分歧义**。



1. 概述

特点：

- 未登录词识别

未登录词又称新词，有如下两种情况：一是词库中没有收录的词，二是训练语料没有出现过的词，主要体现在以下几种：

- (1) 新出现的网络用词：如“蓝牙”等。
- (2) 研究领域名称：特定领域的专有名词，如“禽流感”“三聚氰胺”等。
- (3) 专有名词：如公司企业、专业术语、缩写词等，如“阿里巴巴”“字节跳动”等。



2. 常见中文分词方法

2.1 基于规则和词表方法

基于规则和词表方法的基本思想是基于词典匹配，将待分词的中文文本根据一定规则进行切分，与词典中的词语进行匹配，按照词典中的词语进行分词。代表方法有“正向最大匹配法”“逆向最大匹配法”等。



2. 常见中文分词方法

- 常用词典：
- 搜狗实验室发布的互联网词库（SogouW，其中有15万个词条）
- 清华大学开放中文词库（THUOCL）
- [码农场千万级巨型汉语词库](#)（HanLP 千万级词条）

2. 常见中文分词方法

2.1 基于规则和词表方法

- 完全切分
- 正向最大匹配方法
- 逆向最大匹配方法
- 双向最大匹配方法



2. 常见中文分词方法

- 完全切分指的是，找出一段文本中的所有单词
 - 不是标准意义上的分词

```
def fully_segment(text, dic):  
    word_list = []  
    for i in range(len(text)):  
        for j in range(i + 1, len(text) + 1):  
            word = text[i:j]  
            if word in dic:  
                word_list.append(word)  
    return word_list
```

```
# i从0遍历到text的最后一个字的下标  
# j遍历[i + 1, len(text)]区间  
# 取出连续区间[i, j)对应的字符串  
# 如果在词典中，则认为是一个词
```



2. 常见中文分词方法

正向最大匹配方法(Forward Maximum Matching, FMM)

根据词典中的最长词条所含汉字个数Len, 取出语料中当前位置起始Len个汉字作为查找字符串, 搜索分词词典是否存在匹配词条。如果成功匹配, 完成搜索; 反之, 就取消查找字符串中最后一个汉字, 减少长度, 继续搜索, 直到搜索成功完成这轮匹配任务。重复如上步骤, 直到切分出所有的词。

正向最大匹配分词具有**错误切分率较高, 不能处理交叉歧义和组合歧义**等缺点。



正向最长匹配

```
def forward_segment(text, dic):  
    word_list = []  
    i = 0  
    while i <= len(text):  
        longest_word = text[i] # 扫描当前位置的单字  
        for j in range(i+1, len(text)+1): # 所有可能的结尾  
            word = text[i:j] # 从当前位置到结尾位置的连续字符串  
            if word in dic:  
                if len(word) > len(longest_word): # 越长优先级越高  
                    longest_word = word  
        word_list.append(longest_word) # 输出最长词  
        i += len(longest_word)  
    return word_list
```

2. 常见中文分词方法

2.1 基于规则和词表方法(Forward Maximum Matching, FMM)

【例1】正向最大匹配方法：“研究生命起源”

词典：研究、研究生、生命、起源词典中最长词是“研究生”，其长度是3，从左到右开始取3个字符，步骤如下。

第一步：“研究生”属于词典，因此将“研究生”取出。

第二步：“命起源”不在词典中，长度减1。第三步：“命起”也不在词典中，长度再减1。

第四步：“命”为单字，因此单独取出。

第五步：剩下的“起源”，在词典中。

因此，分词结果为“研究生”“命”和“起源”。



2. 常见中文分词方法

逆向最大匹配法 (Backward Maximum Matching, BMM)。

逆向最大匹配分词过程与正向最大匹配分词方法相同，只是从句子（或者文章）的末尾开始处理。逆向最大匹配法具有**错误切分率，易于处理交叉歧义，但不能处理组合歧义。**



逆向最长匹配

```
def backward_segment(text, dic):  
    word_list = []  
    i = len(text) - 1  
    while i >= 0:  
        longest_word = text[i]  
        for j in range(0, i):  
            word = text[j: i + 1]  
            if word in dic:  
                if len(word) > len(longest_word):  
                    longest_word = word  
        word_list.insert(0, longest_word)  
        i -= len(longest_word)  
    return word_list
```

扫描位置作为终点
扫描位置的单字
遍历[0, i]区间作为待查询词语的起点
取出[j, i]区间作为待查询单词

越长优先级越高

逆向扫描, 因此越先查出的单词在位置上越靠后

表2-1 正向/逆向最长匹配歧义对比

序号	原文	正向最长匹配	逆向最长匹配
1	项目的研究	[项目, 的, 研究]	[项, 目的, 研究]
2	商品和服务	[商品, 和服, 务]	[商品, 和, 服务]
3	研究生命起源	[研究生, 命, 起源]	[研究, 生命, 起源]
4	当下雨天地面积水	[当下, 雨天, 地面, 积水]	[当, 下雨天, 地面, 积水]
5	结婚的和尚未结婚的	[结婚, 的, 和尚, 未, 结婚, 的]	[结婚, 的, 和, 尚未, 结婚, 的]
6	欢迎新老师生前来就餐	[欢迎, 新, 老师, 生前, 来, 就餐]	[欢, 迎新, 老, 师生, 前来, 就餐]



双向最长匹配

- 一种融合两种匹配方法的复杂规则集，流程如下。
 - 同时执行正向和逆向最长匹配，若两者的词数不同，则返回词数更少的那一个。
 - 否则，返回两者中单字更少的那一个。当单字数也相同时，优先返回逆向最长匹配的结果。



双向最长匹配

```
def count_single_char(word_list: list): # 统计单字成词的个数
    return sum(1 for word in word_list if len(word) == 1)
```

```
def bidirectional_segment(text, dic):
    f = forward_segment(text, dic)
    b = backward_segment(text, dic)
    if len(f) < len(b): # 词数更少优先级更高
        return f
    elif len(f) > len(b):
        return b
    else:
        if count_single_char(f) < count_single_char(b): # 单字更少优先级更高
            return f
        else:
            return b # 都相等时逆向匹配优先级更高
```



2. 常见中文分词方法

2.2 基于统计方法

词是稳定的字的组合，因此在上下文中相连的字在不同的文本中出现的次数越多，就证明这些相连的字很可能是一个词，这种利用字与字相邻共现的概率确定词的方法是基于统计的分词方法，一般具有基于隐马尔可夫模型。



2. 常见中文分词方法

2.2 基于统计方法

- 基于隐马尔可夫模型
- 基于最大熵模型
- 基于条件随机场模型
- 基于词网格分词



2. 常见中文分词方法

2.2 基于统计方法

- 基于隐马尔可夫模型

基于隐马尔可夫模型的基本思想是通过文本作为观测序列去确定隐藏序列的过程，该方法采用Viterbi算法对新词识别，但具有生成式模型的缺点。



2. 常见中文分词方法

2.2 基于统计方法

- 基于最大熵模型

基于最大熵模型的基本思想是学习概率模型时，在可能的概率分布模型中。采用最大熵进行切分，该方法可以避免生成模型的不足。但是存在偏移量的问题。



2. 常见中文分词方法

2.2 基于统计方法

- 基于条件随机场模型

基于条件随机场模型的基本思想主要来源于最大熵马尔可夫模型，主要关注的字根与上下文标记位置有关，进而通过解码找到词边界，需要大量训练语料。



2. 常见中文分词方法

2.2 基于统计方法

- 基于词网格分词

基于词网格的基本思想是利用词典匹配，列举输入句子中所有可能的切分词语，并以词网格的形式保存。词网格是一个有向无环图，蕴含输入句子中所有可能的切分，其中的每条路径代表一种切分。根据图搜索算法找出图中权值最优的路径，路径对应的就是最优的分词结果。



2. 常见中文分词方法

2.2 基于理解方法

基于理解的分词方法是通过让计算机模拟人对句子的理解，达到识别词的效果，其基本思想就是在分词的同时进行句法、语义分析，利用句法信息和语义信息来处理歧义现象。

基于理解的分词方法需要使用大量的语言知识和信息。由于汉语语言知识的笼统、复杂性，难以将各种语言信息组织成机器可直接读取的形式，目前基于理解的分词系统还处在实验阶段。



3. 中文分词困惑

采用Sklearn库CountVectorizer函数可以进行英文文章的词频统计。
若文本内容为中文，效果会如何？



3. 中文分词困惑

【例2】中文分词

```
# 例2: 中文分词
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
data = cv.fit_transform(['我来到北京清华大学'])
print('单词数: {}'.format(len(cv.vocabulary_)))
print('分词: {}'.format(cv.vocabulary_))
print(cv.get_feature_names_out())
print(data.toarray())
```

单词数: 1

分词: {'我来到北京清华大学': 0}

['我来到北京清华大学']

[[1]]

程序将整个句子当成了一个词，无法对中文进行分词。英文语句中词与词之间有空格作为天然分隔符，进行分词。而中文却没有。可以将“我来到北京清华大学”通过添加空格进行分隔，变成“我 来到 北京 清华大学”。



3. 中文分词困惑

【例2】中文分词

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
data = cv.fit_transform(['我 来到 北京 清华大学'])
print('单词数: {}'.format(len(cv.vocabulary_)))
print('分词: {}'.format(cv.vocabulary_))
print(cv.get_feature_names_out())
print(data.toarray())
```

单词数: 3

分词: {'来到': 1, '北京': 0, '清华大学': 2}

['北京' '来到' '清华大学']

[[1 1 1]]

这种方法不实用，不可能采用空格进行分词。可以采用jieba和HanLP等分词库进行实现。



4. jieba分词库

4.1 认识jieba

- jieba库是一个python实现的分词库。
- 网址为: <https://github.com/fxsjy/jieba>
- 安装: `pip install jieba`



4. jieba分词库

4.2 三种模式

jieba库支持如下三种分词模式：

- 全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；
- 精确模式，试图将句子最精确地切开，适合文本分析；
- 搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。



4. jieba分词库

4.2 三种模式

【例3】全模式

```
seg_list=jieba.cut(str, cut_all=True)
```

```
# 例3: 全模式
import jieba
seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print("Full mode:"+" ".join(seg_list))
```

Full mode:我/来到/北京/清华/清华大学/华大/大学



4. jieba分词库

4.2 三种模式

【例4】精确模式

```
seg_list=jieba.cut(str, cut_all=False)
```

```
# 例4: 精确模式
import jieba
seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print("Default mode:"+" ".join(seg_list))
```

Default mode:我/来到/北京/清华大学



4. jieba分词库

4.2 三种模式

【例5】搜索引擎模式

seg_list=jieba.cut_for_search(str)

```
# 例5：搜索引擎模式
import jieba
seg_list = jieba.cut_for_search("我来到北京清华大学")
print("/".join(seg_list))
```

我/来到/北京/清华/华大/大学/清华大学



4. jieba分词库

4.3 自定义词典

当基于jieba词库的分词结果不符合需求时，可以通过自定义的词典实现。自定义具有如下两种方式。

方式1：词典文件

方法2：动态修改词频



4. jieba分词库

4.3 自定义词典

方式1：词典文件

通过添加词典文件定义分词的最小单位，文件需要有特定格式，并且为UTF-8编码。

```
jieba.load_userdict(file_name)    # file_name为自定义词典
```



4. jieba分词库

4.3 自定义词典

方式1：词典文件

【例6】`jieba.load_userdict(file_name)`

```
# 例6: jieba.load_userdict举例
import jieba
seg_list = jieba.cut("周元哲老师是Python技术讲师", cut_all = True)
print("/".join(seg_list))
```

周/元/哲/老师/是/Python/技术/讲师

周 / 元 / 哲 / 被分隔为“周”“元”和“哲”，不符合开发者的预期。添加自定义词典，在C:\Users\自然语言处理\userdict.txt文件，内容遵守如下规则：一个词占一行；每一行分为三部分：词语、词频（可省略），词性（可省略），用空格隔开，顺序不可颠倒。本例的userdict.txt文件内容为：周元哲 3 n。

词典案例

希望	v	386	n	96
希罕	a	1		
希翼	v	1		
希腊	ns	19		
希腊共和国	ns	1		

MacBook pro,n,5



4. jieba分词库

4.3 自定义词典

方式1: 词典文件

【例6】 `jieba.load_userdict(file_name)`

```
import jieba
import os
jieba.load_userdict("C:\\Users\\自然语言处理\\userdict.txt")
seg_list = jieba.cut("周元哲老师是Python技术讲师", cut_all = True)
print("/".join(seg_list))
```

周元哲/老师/是/Python/技术/讲师



4. jieba分词库

4.3 自定义词典

方式2：动态修改词频

调节单个词语的词频，使其能（或者不能）被划分出来。

```
jieba.suggest_freq(segment, tune=True)
```



4. jieba分词库

4.3 自定义词典

方式2：动态修改词频

【例7】jieba.suggest_freq举例

```
# 例7: jieba.suggest_freq举例
import jieba
jieba.suggest_freq("周元哲", tune = True)
seg_list = jieba.cut("周元哲老师是Python技术讲师", cut_all=True)
print("/".join(seg_list))
```

周元哲/老师/是/Python/技术/讲师



4. jieba分词库

4.4 词性标注

词性标注指为分词结果中的每个词标注正确的词性，即确定每个词是名词、动词、形容词或其他词性的过程。例如，“周元哲”是名词，“是”是动词，“老师”是名词等。

词性标注有多个重要作用。

第一，消除歧义。一些词在不同语境或不同用法时表示不同的意思。例如，在“这只狗的名字叫开心”和“我很开心”这两个句子中，“开心”代表了不同的含义，可以通过词性标注进行区分。

第二，强化基于单词的特征。如果不进行词性标注，两个“开心”会被认为是同义词，在后续分析中引入误差。



4. jieba分词库

4.4 词性标注

jieba.posseg.cut()

【例8】词性标注

```
# 例8: 词性标注
import jieba.posseg as pseg
words = pseg.cut("周元哲老师是Python技术讲师")
for word,flag in words:
    print("%s%s"%(word,flag))
```

周元哲x
老师n
是v
Pythoneng
技术n
讲师n



4. jieba分词库

4.4 词性标注

表4-1 常见词性表

词性	描述	词性	描述	词性	描述	词性	描述
Ag	形容素	G	语素	ns	地名	z	状态词
a	形容词	H	前接成分	nt	机构团体	W	标点符号
ad	副词	l	成分	nz	其他专名	v	动词
an	名形词	J	简称略语	o	拟声词	y	语气词
b	区别词	K	后接成分	p	介词	Vn	名动词
c	连词	L	习用语	q	量词	vg	动语素
dg	副语素	M	数词	r	代词	x	非语素字
d	副词	Ng	名语素	s	处所词	Vd	副动词
e	叹词	N	名词	tg	时语速	u	助词
f	方位词	Nr	人名	t	时间词		



4. jieba分词库

4.5 断词位置

断词位置用于返回每个分词的起始和终止位置

jieba.Tokenizer()

【例9】断词位置

```
# 例9: 断词位置
import jieba
result = jieba.tokenize('周元哲老师是Python技术讲师') # 返回词语在原文的起止位置
print("默认模式为: ")
for tk in result:
    print("word %s\t\t start: %d \t\t end:%d" % (tk[0],tk[1],tk[2]))
```

默认模式为:

word 周元哲	start: 0	end:3
word 老师	start: 3	end:5
word 是	start: 5	end:6
word Python	start: 6	end:12
word 技术	start: 12	end:14
word 讲师	start: 14	end:16



4. jieba分词库

4.6 关键词抽取

基于TF-IDF算法计算文本中词语的权重，进行关键词提取

```
jieba.analyse.extract_tags(lines, topK=20, withWeight=False, allowPOS=())
```

参数	说明
lines	待提取的文本
topK	返回TF/IDF权重最大的关键词的个数，不超过20个
withWeight	是否一并返回关键词权重，默认值为False
allPOS	词性过滤，默认值为空，表示不过滤

表4-1 jieba.analyse.extract_tags()参数说明



4. jieba分词库

4.6 关键词抽取

基于TF-IDF算法计算文本中词语的权重，进行关键词提取

```
jieba.analyse.extract_tags(lines, topK=20, withWeight=False, allowPOS=())
```

【例10】基于TF-IDF算法的关键词抽取

```
# 例10: 基于TF-IDF的关键词抽取
import jieba.analyse as analyse
lines = "周元哲老师是Python技术讲师"
keywords = analyse.extract_tags(lines, topK=20, withWeight = True, allowPOS=())
for item in keywords:
    print("%s = %f"%(item[0],item[1]))
```

```
周元哲 = 2.390954
Python = 2.390954
讲师 = 1.727597
老师 = 1.274684
技术 = 0.943891
```



4. jieba分词库

4.6 关键词抽取

【例10】基于TF-IDF算法的关键词抽取

jieba给每一个分词标出IDF分数比重，可以通过设定IDF分数高或低。突出或降低某关键字的权重。jieba的IDF分数一般为9到12，自定义IDF分数为2到5。

创建自定比重分数文件，在C:\Users\自然语言处理\idf.txt文件，内容遵守如下规则：一个词占一行；每一行分为两部分：词语、权重，用空格隔开，顺序不可颠倒，文件为UTF -8编码格式。本例idf.txt 文件内容如下。

周元哲 5

讲师 9



4. jieba分词库

4.6 关键词抽取

【例11】自定义比重分数

```
# 例11: 自定义比重分数
import jieba
import jieba.analyse as analyse
lines = "周元哲老师是Python技术讲师"
print('default idf' + '-' * 90)
keywords = analyse.extract_tags(lines, topK = 10, withWeight = True, allowPOS = ())
for item in keywords:
    print("%s = %f"%(item[0],item[1]))

print('set_idf_path' + '-' * 90)
jieba.analyse.set_idf_path("C:\\Users\\Administrator\\自然语言处理\\idf.txt")
keywords = analyse.extract_tags(lines, topK = 10, withWeight = True, allowPOS = ())
# print("topK = TF/IDF, TF = %d"%len(keywords))
for item in keywords:
    # print("%s = %f"%(item[0],item[1]))
    print("%s TF = %f, IDF = %f topK = %f"%(item[0],item[1],len(keywords)*item[1],item[1]*len(keywords)*item[1]))
```



4. jieba分词库

4.6 关键词抽取

【例11】 自定义比重分数

```
default idf-----
周元哲 = 2.390954
Python = 2.390954
讲师 = 1.727597
老师 = 1.274684
技术 = 0.943891
set_idf_path-----
老师 TF =1.800000,IDF =9.000000 topK= 16.200000
Python TF =1.800000,IDF =9.000000 topK= 16.200000
技术 TF =1.800000,IDF =9.000000 topK= 16.200000
讲师 TF =1.800000,IDF =9.000000 topK= 16.200000
周元哲 TF =1.000000,IDF =5.000000 topK= 5.000000
```



4. jieba分词库

4.6 关键词抽取

将每个分词当成key，将其在文中出现的次数作为value，最后进行降序排列，即可排列出最常出现的分词。

【例12】排列最常出现的分词

```
# 例12: 排列最常出现的分词
import jieba
text = "周元哲老师是Python技术讲师,周元哲老师是软件测试技术讲师"
dic = {}
for ele in jieba.cut(text):
    if ele not in dic:
        dic[ele] = 1
    else:
        dic[ele] = dic[ele] + 1
for w in sorted(dic, key = dic.get, reverse=True):
    print("%s %i"%(w,dic[w]))
```

周元哲 2
老师 2
是 2
技术 2
讲师 2
Python 1
, 1
软件测试 1



4. jieba分词库

4.7 停用词表

如果只使用词频来衡量重要性，很容易过度强调在文档中经常出现而并没有包含太多与文档有关的信息的词语，如文档中的标点、空格、没有意义的字-“的，了……”等。停用词表（Stop Words）将这些没有什么实际含义的功能词汇总，可在网上下载，如网址

<https://github.com/goto456/stopwords>，命名
StopWords.txt。



4. jieba分词库

4.7 停用词表

【例13】 使用jieba分析刘慈欣小说《三体》中出现次数最多的词语，《三体》小说保存在C:\\Users\\自然语言处理\\Three_Body.txt，文件为UTF-8编码

```
# 例14: 停用词
import jieba
txt = open("C:\\Users\\自然语言处理\\Three_Body.txt", encoding="utf-8").read()
words = jieba.lcut(txt)
counts = {}
for word in words:
    counts[word] = counts.get(word,0) + 1
items = list(counts.items())
items.sort(key=lambda x:x[1], reverse=True)
for i in range(30):
    word, count = items[i]
    print ("{:<10}{1:>5}".format(word, count))
```

的	36269
,	27042
	21805
。	19428
	18956
	18885
, 了	10170
“	8705
”	8591
在是	8382
	7003
	6841
他	4208
中	3659
我	3362
和	3215
一个	3057
都	2959
上	2798
她	2751
说	2748
这	2724
你	2707
也	2672
但	2602
有	2507
着	2270
就	2234
不	2202
没有	2128

4. jieba分词库

4.7 停用词表

【例14】使用停用词，利用jieba分析刘慈欣小说《三体》中出现次数最多的词语

```
import jieba
txt = open("C:\\Users\\自然语言处理\\Three_Body.txt", encoding="utf-8").read()
#加载停用词表
stopwords = [line.strip() for line in open("C:\\Users\\自然语言处理\\stopwords-master\\hit_stopwords.txt", encoding="utf-8").readlines()]
words = jieba.lcut(txt)
counts = {}
for word in words:
    #不在停用词表中
    if word not in stopwords:
        #不统计字数为一的词
        if len(word) == 1:
            continue
        else:
            counts[word] = counts.get(word,0) + 1
items = list(counts.items())
items.sort(key=lambda x:x[1], reverse=True)
for i in range(30):
    word, count = items[i]
    print ("<10>{:>7}".format(word, count))
```

《自然语言处理入门》

没有	2128
程心	1320
现在	1273
已经	1259
世界	1243
罗辑	1189
可能	1177
看到	1114
知道	1094
地球	951
人类	935
太空	930
三体	883
宇宙	875
太阳	775
不是	726
舰队	649
飞船	644
这种	638
出现	627
时间	611
汪淼	606
两个	580
文明	561
最后	541
起来	525
东西	515
发现	500
进行	495
这是	491



4. jieba分词库

4.7 停用词表

【例15】引入jieba和停止词，进行中文特征提取

```
# 例15: 引入jieba和停用词, 进行中文特征提取
from sklearn.feature_extraction.text import CountVectorizer
import jieba
text = '今天天气真好, 我要去西安大雁塔玩, 玩完之后, 游览兵马俑'
# 进行结巴分词, 精确模式
text_list = jieba.cut(text, cut_all=False)
text_list = ",".join(text_list)
context = []
context.append(text_list)
print(context)

con_vec = CountVectorizer(min_df=1, stop_words=['之后', '玩完'])
X = con_vec.fit_transform(context)
feature__name = con_vec.get_feature_names_out()
print(feature__name)
print(X.toarray())
```

```
['今天天气,真好,,我要,去,西安,大雁塔,玩,,玩完,之后,,游览,兵马俑']
['今天天气' '兵马俑' '大雁塔' '我要' '游览' '西安']
[[1 1 1 1 1 1]]
```



5. HanLP分词

5.1 认识HanLP

HanLP提供如下功能：

(1) 中文分词

1. 最短路分词；2. N-最短路分词；3. CRF分词；4. 索引分词；5. 极速词典分词；6. 用户自定义词典；7. 标准分词

(2) 命名实体识别

1. 实体机构名识别；2. 中国人名识别；3. 音译人名识别；4. 日本人名识别；5. 地名识别

(3) 篇章理解

1. 关键词提取；2. 自动摘要；3. 短语提取

(4) 简繁拼音转换

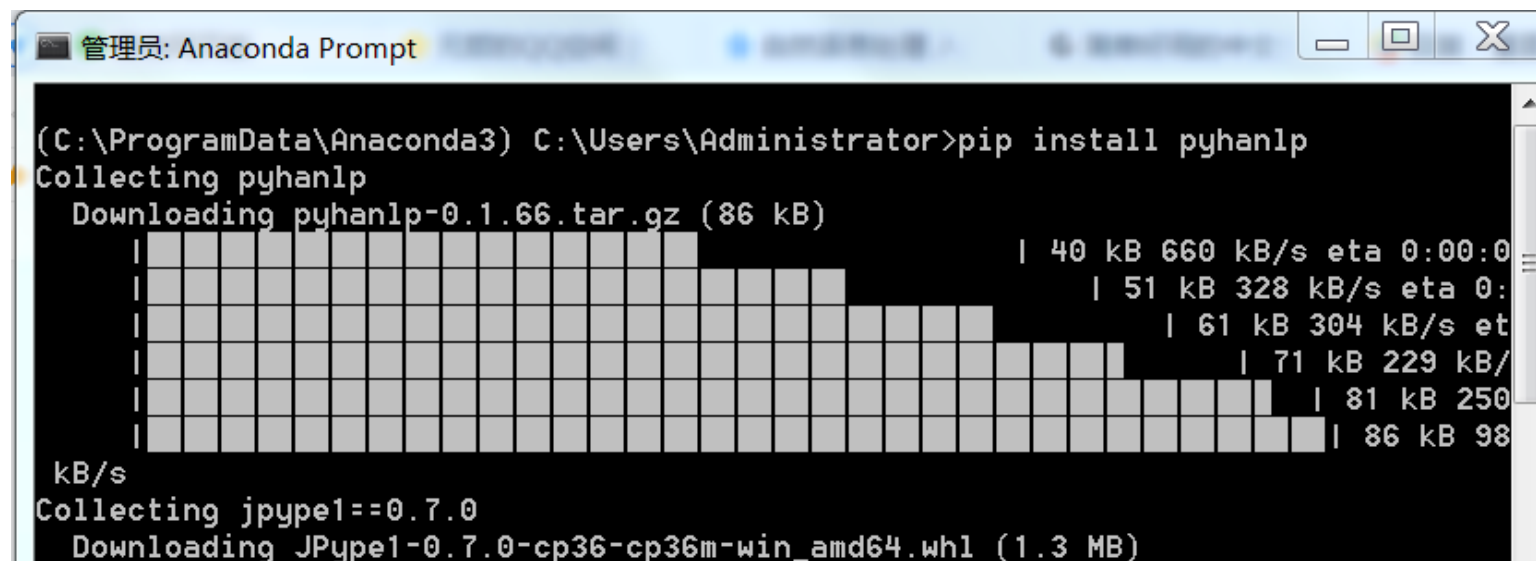
1. 拼音转换；2. 简繁转换



5. HanLP分词

5.2 pyhanlp

pyhanlp是HaLP的Python接口，使用Python可以访问操作HanLP。在Anaconda下的命令提示符下输入`pip install pyhanlp`进行安装。



```
(C:\ProgramData\Anaconda3) C:\Users\Administrator>pip install pyhanlp
Collecting pyhanlp
  Downloading pyhanlp-0.1.66.tar.gz (86 kB)
    | 40 kB 660 kB/s eta 0:00:00
    | 51 kB 328 kB/s eta 0:
    | 61 kB 304 kB/s et
    | 71 kB 229 kB/
    | 81 kB 250
    | 86 kB 98
  kB/s
Collecting jpyype1==0.7.0
  Downloading JPyype1-0.7.0-cp36-cp36m-win_amd64.whl (1.3 MB)
```



5. HanLP分词

5.3 中文分词

HanLP提供segment()方法给出每句文本的分词

【例16】中文分词

```
# 例16: 中文分词
#coding=utf-8
from pyhanlp import *
print(HanLP.segment('你好，欢迎在Python中调用 HanLP的API'))
```

[你好/vl, , /w, 欢迎/v, 在/p, Python/nx, 中/f, 调用/v, /w, HanLP/nx, 的/ude1, API/nx]

可见，hanlp的分词结果是带有词性的



5. HanLP分词

5.4 依存分析使用

HanLP提供parseDependency()方法进行每句文本中词语之间的相互依存关系

【例17】依存分析使用

```
# 例17: 依存句法分析
from pyhanlp import *
print(HanLP.parseDependency("今天开心了吗? "))
```

1	今天	今天	nt	t	—	2	状中结构	—	—
2	开心	开心	a	a	—	0	核心关系	—	—
3	了	了	e	y	—	2	右附加关系	—	—
4	吗	吗	e	y	—	2	右附加关系	—	—
5	?	?	wp	w	—	2	标点符号	—	—



5. HanLP分词

5.4 依存分析使用

pyhanlp提供了友好的展示交互页面，使用命令hanlp serve启动一个web服务，登录 <http://localhost:8765>出现可视化界面，直观展示分词结果、依存关系，并给出安装说明、源码链接、文档链接、常见的问题（FAQ）等信息。



5. HanLP分词

5.4 依存分析使用

```
(base) PS C:\Users\Administrator> conda activate pyhan38  
(pyhan38) PS C:\Users\Administrator> hanlp serve  
服务器已启动 http://localhost:8765
```

尝试输入“今天开心了吗？”



5. HanLP分词

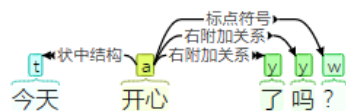
5.4 依存分析使用



5. HanLP分词

5.4 依存分析使用

句法分析



5. HanLP分词

5.5 关键词提取

HanLP提供extractKeyword()方法进行关键词提取

【例18】关键词提取

```
# 例18: 关键词提取
from pyhanlp import *
content = (
    "程序员(英文Programmer)是从事程序开发、维护的专业人员。"
    "一般将程序员分为程序设计人员和程序编码人员，"
    "但两者的界限并不非常清楚，特别是在中国。"
    "软件从业人员分为初级程序员、高级程序员、系统"
    "分析员和项目经理四大类。")
TextRankKeyword = JClass("com.hankcs.hanlp.summary.TextRankKeyword")
keyword_list = HanLP.extractKeyword(content, 5)
print(keyword_list)
# print(help(HanLP))
```



5. HanLP分词

5.6 命名实体识别

HanLP提供enableNameRceognize()方法进行命名实体识别

【例19】人名识别

```
# 例19: 人名识别
from pyhanlp import *
NER=HanLP.newSegment().enableNameRecognize(True)
p_name=NER.seg('马云、雷军、汪洋、张朝阳的搜狗、韩寒的书、马化腾的腾讯')
print(p_name)
```

[马云/nr, 、/w, 雷军/nr, 、/w, 汪洋/n, 、/w, 张朝阳/nr, 的/ude1, 搜狗/gi, 、/w, 韩寒/nr, 的/ude1, 书/n, 、/w, 马化腾/nr, 的/ude1, 腾讯/ntc]



5. HanLP分词

5.7 自定义词典

HanLP提供CustomDictionary进行自定义词典

【例20】自定义词典

```
# 例20: 自定义词典
from pyhanlp import *
content = "铁甲网是中国最大的工程机械交易平台"
print(HanLP.segment(content))
CustomDictionary.add("铁甲网")
CustomDictionary.insert("工程机械", "nz 1024")
CustomDictionary.add("交易平台", "nz 1024 n 1")
print(HanLP.segment(content))
```

[铁甲/n, 网/n, 是/vshi, 中国/ns, 最大/gm, 的/ude1, 工程机械/nz, 交易平台/nz]

[铁甲网/nz, 是/vshi, 中国/ns, 最大/gm, 的/ude1, 工程机械/nz, 交易平台/nz]



5. HanLP分词

5.8 简繁体转换

HanLP提供convertToSimplifiedChinese() 和 convertToTraditionalChinese() 方法进行简繁体转换

【例21】简繁体转换

```
# 例21: 简繁体转换
from pyhanlp import *

Jianti = HanLP.convertToSimplifiedChinese("我愛自然語言處理技術!")
Fanti = HanLP.convertToTraditionalChinese("我爱自然语言处理技术!")
print(Jianti)
print(Fanti)
```

我爱自然语言处理技术!
我愛自然語言處理技術!



5. HanLP分词

5.9 摘要提取

HanLP提供extractSummary()方法进行摘要提取

【例22】摘要提取

```
# 例22: 摘要提取
from pyhanlp import *
document = '''自然语言处理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。因此，这一领域的研究将涉及自然语言，即人们日常使用的语言，所以它与语言学的研究有着密切的联系，但又有重要的区别。自然语言处理并不是一般地研究自然语言，而在于研制能有效地实现自然语言通信的计算机系统，特别是其中的软件系统。因而它是计算机科学的一部分。'''
TextRankSentence = JClass("com.hankcs.hanlp.summary.TextRankSentence")
sentence_list = HanLP.extractSummary(document, 3)
print(sentence_list)
sentence_list = HanLP.extractSummary(document, 2)
print(sentence_list)
sentence_list = HanLP.extractSummary(document, 1)
print(sentence_list)
sentence_list = HanLP.getSummary(document, 50)
print(sentence_list)
sentence_list = HanLP.getSummary(document, 30)
print(sentence_list)

sentence_list = HanLP.getSummary(document, 20)
print(sentence_list)
```

[自然语言处理并不是一般地研究自然语言，自然语言处理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]
[自然语言处理并不是一般地研究自然语言，自然语言处理是计算机科学领域与人工智能领域中的一个重要方向]
[自然语言处理并不是一般地研究自然语言]
自然语言处理是计算机科学领域与人工智能领域中的一个重要方向。自然语言处理并不是一般地研究自然语言。
自然语言处理是计算机科学领域与人工智能领域中的一个重要方向。
自然语言处理并不是一般地研究自然语言。



