Natural Language Processing I

4. 词典分词

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

2022/03/18

词、词典

分，还是不分？

小鸡炖蘑菇。

小鸡 炖 蘑菇。

HanLP 在线分析

为什么要分词

- 语义理解依赖于正确的分词
  - 词：语义的最小单位
  - 现代汉语
- 很多 NLP 算法依赖于分词

为什么要分词

- 语义理解依赖于正确的分词
  - 词：语义的最小单位
  - 现代汉语
- 很多 NLP 算法依赖于分词

两大类分词算法

- 词典规则：约定俗成或专家指定
- 机器学习：从数据本身进行分析

词

语言学定义：具备**独立语义的最小单元**

- 这个定义没有实际应用价值
  - 什么叫“最小单元”?

词

语言学定义：具备**独立语义的最小单元**

- 这个定义没有实际应用价值
  - 什么叫“最小单元”?

例如：小明的妈妈喊他回家吃饭

```
吃饭!
吃什么饭?
小鸡炖蘑菇。
谁炖的蘑菇？！
```

词

语言学定义：具备**独立语义的最小单元**

- 这个定义没有实际应用价值
  - 什么叫“最小单元”?

例如：小明的妈妈喊他回家吃饭

```
吃饭!
吃什么饭?
小鸡炖蘑菇。
谁炖的蘑菇？！
```

注意：汉字本身具有实意，特别是古汉语

```
妻子
```

词

语言学定义：具备**独立语义的最小单元**

- 这个定义没有实际应用价值
  - 什么叫“最小单元”?

例如：小明的妈妈喊他回家吃饭

```
吃饭!
吃什么饭?
小鸡炖蘑菇。
谁炖的蘑菇？！
```

注意：汉字本身具有实意，特别是古汉语

```
妻子
```

```
“玷污……玷是一个动作，污是一个结果” -- 李敖在北大的演讲
```

词典分词

现实语言中，词汇和语法是不断发展变化的

- 需要一个**相对稳定的参考**

词典分词

现实语言中，词汇和语法是不断发展变化的

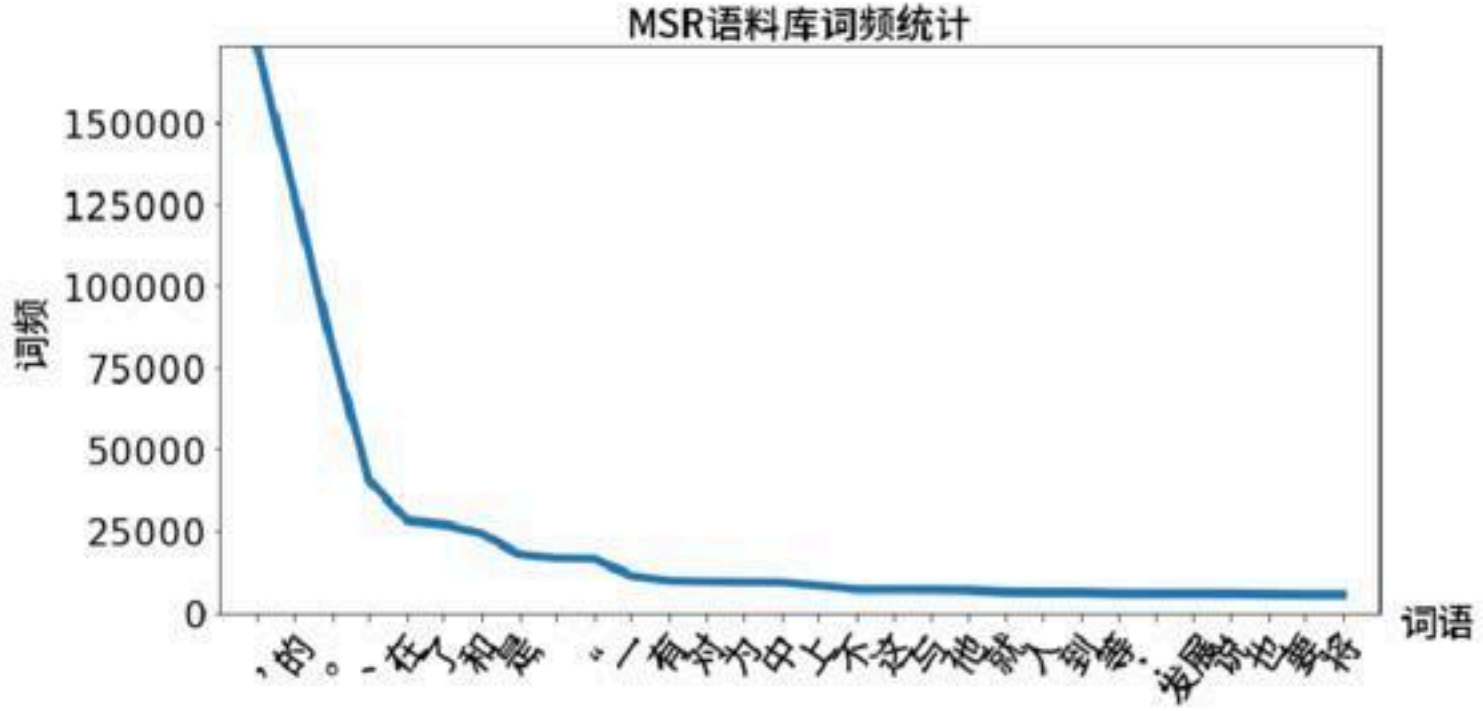
- 需要一个**相对稳定的参考**

词典分词：只认词典里面的字符串

- 谁来编制词典?
- 如何更新词典?

Zipf 定律

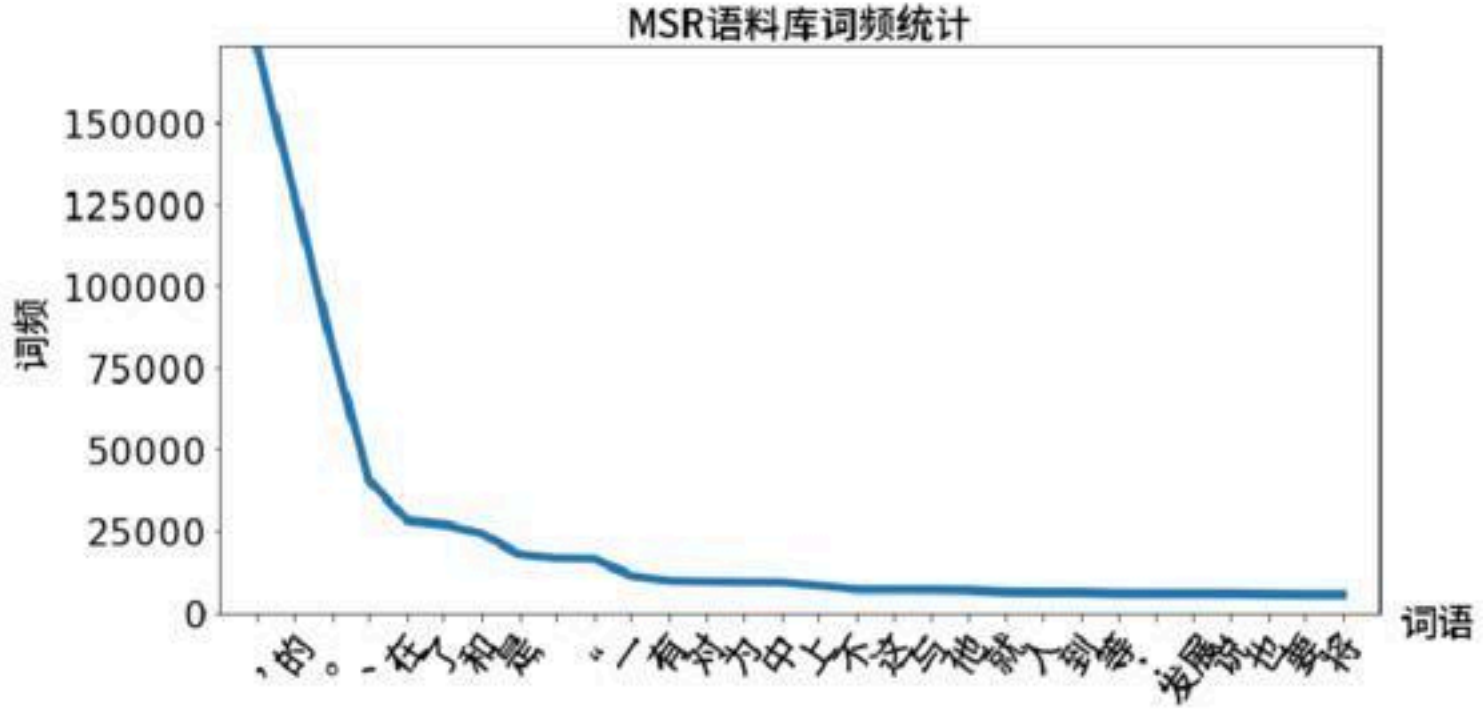
一个单词的词频与其词频排名成反比



- MSR 语料库

Zipf 定律

一个单词的词频与其词频排名成反比



- MSR 语料库
- 大致符合幂律分布（长尾效应）：罕见词很多但影响不大

HanLP 词典

```
希望  v   386 n   96  vn  25  nz  1
希特勒 nr  3
希玛  nz  1
希罕  a   1
希翼  v   1
希腊  ns  19
```

- 以空格分隔
- 每两列：词类+词频

完全切分

找出所有可能的单词

- 遍历连续子序列，查询是否在词典中

```
商品和服务
['商', '商品', '品', '和', '和服', '服', '服务', '务']
```

完全切分

找出所有可能的单词

- 遍历连续子序列，查询是否在词典中

```
商品和服务
['商', '商品', '品', '和', '和服', '服', '服务', '务']
```

- 注意：这**不是**标准意义上的分词

完全切分：实现

```
word_list = []
for i in range(len(text)):                          # i 从 0 到text的最后一个字的下标遍历
    for j in range(i + 1, len(text) + 1):           # j 遍历[i + 1, len(text)]区间
        word = text[i:j]                            # 取出连续区间[i, j]对应的字符串
        if word in dic:                             # 如果在词典中，则认为是一个词
```

```
商品和服务
['商', '商品', '品', '和', '和服', '服', '服务', '务']
```

正向最长匹配

我们需要将文本分隔成**有意义的词语序列**

- “中华人民共和国”

正向最长匹配

我们需要将文本分隔成**有意义的词语序列**

- “中华人民共和国”

最长匹配：优先输出更长的词汇

- 经验规律：一般越长的词汇表达的意义越丰富
- 正向：从前往后

```
中华人民共和国是我的祖国
['中华人民共和国', '是', '我', '的', '祖国']
```

正向最长匹配：实现

```
word_list = []
i = 0
while i < len(text):
    longest_word = text[i]                          # 当前扫描位置的单字
    for j in range(i + 1, len(text) + 1):           # 所有可能的结尾
        word = text[i:j]                            # 从当前位置到结尾的
        连续字符串
        if word in dic:                             # 在词典中
            if len(word) > len(longest_word):       # 并且更长
```

```
中华人民共和国是我的祖国
['中华人民共和国', '是', '我', '的', '祖国']
```

正向最长匹配：问题

```
研究自然语言处理
['研究', '自然', '语言', '处理']
```

- 将专有名词加入词典

正向最长匹配：问题

```
研究自然语言处理
['研究', '自然', '语言', '处理']
```

- 将专有名词加入词典

```
研究生命起源
['研究生', '命', '起源']
```

- 既然正向不行，那就逆向匹配

逆向最长匹配：实现

```
word_list = []
i = len(text) - 1
while i >= 0:                                    # 扫描位置作为终点
    longest_word = text[i]                       # 扫描位置的单字
    for j in range(0, i):                        # 遍历[0, i]区间作
        为待查询词语的起点
        word = text[j: i + 1]                    # 取出[j, i]区间作
        为待查询单词
        if word in dic:
```

```
研究生命起源
['研究', '生命', '起源']
```

逆向最长匹配

既然正向不行，那就逆向匹配

- 实践总结的规律

```
研究生命起源
['研究', '生命', '起源']
```

逆向最长匹配

既然正向不行，那就逆向匹配

- 实践总结的规律

```
研究生命起源
['研究', '生命', '起源']
```

- 但也不能解决所有问题

```
项目的研究
['项', '目的', '研究']
```

最长匹配的歧义对比

原文	正向	逆向
项目的研究	**['项目', '的', '研究']**	['项', '目的', '研究']
商品和服务	['商品', '和服', '务']	**['商品', '和', '服务']**
研究生命起源	['研究生', '命', '起源']	**['研究', '生命', '起源']**
当下雨天地面积水	['当下', '雨天', '地面', '积水']	**['当', '下雨天', '地面', '积水']**
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']**
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']

- 逆向匹配似乎更好
- 存在两种方法都不能消除歧义的情况

双向最长匹配

融合两个方向匹配的复杂规则集（即专家系统）

1. 若两者返回词数不同，取**词数较少**的那个
2. 若相同，取**单字更少**的那个
3. 否则**优先返回逆向**最长匹配

语言学上的启发：也称“**启发式算法**”

- （现代）汉语中单字词远少于多字词
- 逆向最长匹配确实表现要好一些

双向最长匹配：实现

```
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:
```

双向最长匹配：实现

```
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:
```

- 规则集（专家系统）的维护非常麻烦：**无限的打补丁**

双向最长匹配：效果

原文	正向	逆向	双向
项目的研究	**['项目', '的', '研究']**	['项', '目的', '研究']	['项', '目的', '研究']
商品和服务	['商品', '和服', '务']	**['商品', '和', '服务']**	**['商品', '和', '服务']**
研究生命起源	['研究生', '命', '起源']	**['研究', '生命', '起源']**	**['研究', '生命', '起源']**
当下雨天地面积水	['当下', '雨天', '地面', '积水']	**['当', '下雨天', '地面', '积水']**	['当下', '雨天', '地面', '积水']
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']**	**['结婚', '的', '和', '尚未', '结婚', '的']**
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']	['欢', '迎新', '老', '师生', '前来', '就餐']

双向最长匹配：效果

原文	正向	逆向	双向
项目的研究	**['项目', '的', '研究']**	['项', '目的', '研究']	['项', '目的', '研究']
商品和服务	['商品', '和服', '务']	**['商品', '和', '服务']**	**['商品', '和', '服务']**
研究生命起源	['研究生', '命', '起源']	**['研究', '生命', '起源']**	**['研究', '生命', '起源']**
当下雨天地面积水	['当下', '雨天', '地面', '积水']	**['当', '下雨天', '地面', '积水']**	['当下', '雨天', '地面', '积水']
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']**	**['结婚', '的', '和', '尚未', '结婚', '的']**
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']	['欢', '迎新', '老', '师生', '前来', '就餐']

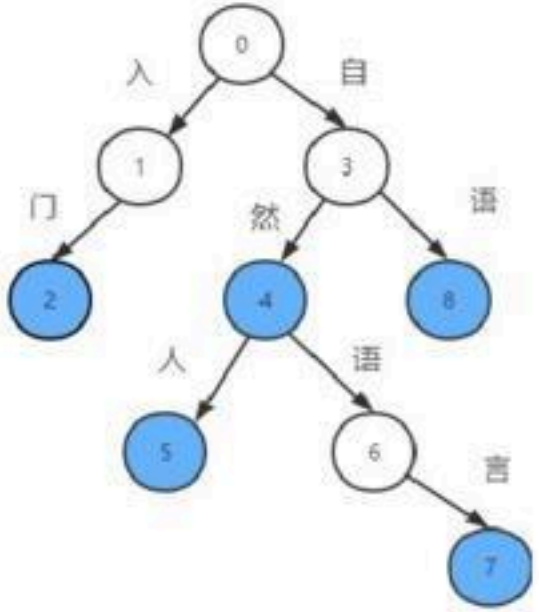
- 违反"**Occam剃刀原则**"：复杂，但还没逆向匹配效果好

用于分词的数据结构

字典树

字典树 trie：也称前缀树，是用于处理字符串的树形数据结构

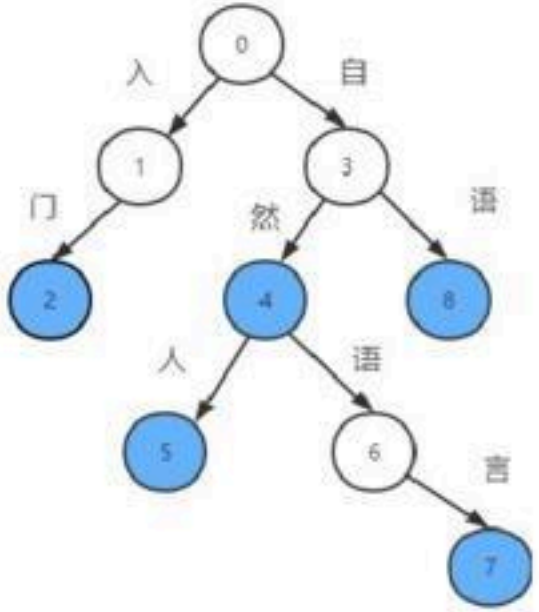
- 每条边代表一个字（符）
- 每条路径构成一个字符串



字典树

字典树 trie：也称前缀树，是用于处理字符串的树形数据结构

- 每条边代表一个字（符）
- 每条路径构成一个字符串



- 数字只是人为编号
- 词汇终点添加终止标记
  - 注意：不一定是叶节点

字典树：节点实现

```
class Node(object):
    def __init__(self, value) -> None: # Type Hints: returns
        nothing
        self._children = {}
        self._value = value

    def _add_child(self, char, value, overwrite=False):
        child = self._children.get(char)
        if child is None: # None: 当前节点不对应任何词
```

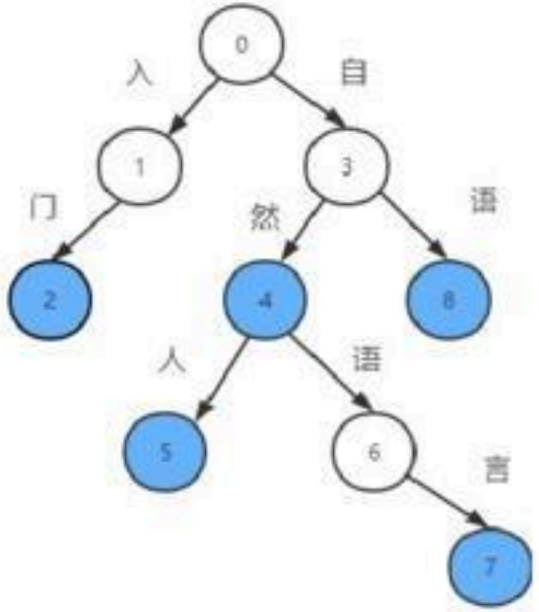
字典树：增删改查

“增删改查”的关键是查询

- **删除**：将终点设为None
- **修改**：将终点替换

查询：二叉树的遍历

- **增加**：无法遍历时，创建子节点



字典树：增删改查实现

```
class Trie(Node):
    def __init__(self) -> None:
        super().__init__(None)

    def __contains__(self, key):
        return self[key] is not None

    def __getitem__(self, key):
        state = self
```

散列表

问题：**起点（根节点）怎么处理?**

- 给一句话中每个字都创建`trie`?

```
北京大学位于北京市
```

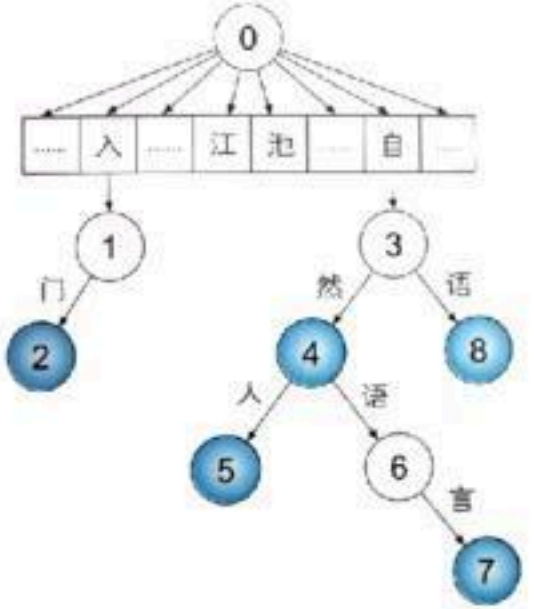
散列表

问题：**起点（根节点）怎么处理?**

- 给一句话中每个字都创建trie?

```
北京大学位于北京市
```

- 首字（根节点）散列表
  - 散列查找
- 其余节点按trie构造
  - 二分查找



其他数据结构

提升匹配速度

- 双数组字典树 DAT
- AC (Aho-Corasick) 自动机

其他数据结构

提升匹配速度

- 双数组字典树 DAT
- AC (Aho-Corasick) 自动机

分词问题本身不太重要

- NLP的最终目的是语义理解，分词只是**可选预处理步骤**
  - 提升速度对预处理来说只是锦上添花
  - [Li 2019] 中文用汉字作为基本单元可以解决大多数NLP任务

其他数据结构

提升匹配速度

- 双数组字典树 DAT
- AC (Aho-Corasick) 自动机

分词问题本身不太重要

- NLP的最终目的是语义理解，分词只是**可选预处理步骤**
  - 提升速度对预处理来说只是锦上添花
  - [Li 2019] 中文用汉字作为基本单元可以解决大多数NLP任务

词典分词的局限性

- 现代分词算法更加强调**数据本身的结构信息**
  - 主要应用机器学习；词典作为辅助信息

Demo：HanLP 词典分词

评测

评测词典分词

混淆矩阵：预测与标注数量相等

	T	F	
P	TP	FP	$P = \frac{TP}{TP+FP}$
N	FN	TN	
	$R = \frac{TP}{TP+FN}$		

评测词典分词

混淆矩阵：预测与标注数量相等

	T	F	
P	TP	FP	$P = \frac{TP}{TP+FP}$
N	FN	TN	
	$R = \frac{TP}{TP+FN}$		

例如：“结婚的和尚未结婚的”

- 标注：['结婚', '的', '和', '尚未', '结婚', '的']
- 预测：['结婚', '的', '和尚', '未结婚', '的']

分隔转换

	T	F	
P	TP	FP	$P = \frac{TP}{TP+FP}$
N	FN	TN	
	$R = \frac{TP}{TP+FN}$		

例如："结婚的和尚未结婚的"

- 标注：['结婚', '的', '和', '尚未', '结婚', '的']
- 预测：['结婚', '的', '和尚', '未结婚', '的']

需要将分隔**转换成分类**

- 将每个词的起止位置记作区间$[w_i, w_j]$
  - **标注真值**对应的区间集合记作A：$A = TP \cup FN$
  - **预测阳性**对应的区间集合记作B：$B = TP \cup FP$

分隔转换

	T	F	
P	TP	FP	$P = \frac{TP}{TP+FP}$
N	FN	TN	
	$R = \frac{TP}{TP+FN}$		

例如："结婚的和尚未结婚的"

- 标注：['结婚', '的', '和', '尚未', '结婚', '的']
- 预测：['结婚', '的', '和尚', '未结婚', '的']

需要将分隔**转换成分类**

- 将每个词的起止位置记作区间$[w_i, w_j]$
  - **标注真值**对应的区间集合记作A：$A = TP \cup FN$
  - **预测阳性**对应的区间集合记作B：$B = TP \cup FP$

推论：$TP = A \cap B$

- $P = \frac{|A \cap B|}{|B|}, R = \frac{|A \cap B|}{|A|}$

分隔转换：举例

例如："结婚的和尚未结婚的"

	单词序列	集合	集合元素
标注	['结婚', '的', '和', '尚未', '结婚', '的']	A	$[1,3],[3,4],[4,5],[5,7],[7,9],[9,10]$
预测	['结婚', '的', '和尚', '未结婚', '的']	B	$[1,3],[3,4],[4,6],[6,9],[9,10]$
重合	['结婚', '的', ~~'和尚未结婚'~~, '的']	$A \cap B$	$[1,2],[3,4],[9,10]$

分隔转换：举例

例如：“结婚的和尚未结婚的”

	单词序列	集合	集合元素
标注	[‘结婚’, ‘的’, ‘和’, ‘尚未’, ‘结婚’, ‘的’]	A	$[1,3],[3,4],[4,5],[5,7],[7,9],[9,10]$
预测	[‘结婚’, ‘的’, ‘和尚’, ‘未结婚’, ‘的’]	B	$[1,3],[3,4],[4,6],[6,9],[9,10]$
重合	[‘结婚’, ‘的’, ~~‘和尚未结婚’~~, ‘的’]	$A \cap B$	$[1,2],[3,4],[9,10]$

- $P = \frac{|A \cap B|}{|B|} = \frac{3}{5} = 60\%, R = \frac{|A \cap B|}{|A|} = \frac{3}{6} = 50\%$
- $F_1 = \frac{2PR}{P+R} = 55\%$

评测：词典选用

评测与**词典粒度**

- 粒度较粗：收录的词较长，如“圆满完成”
  - 算法与评测应该使用**相同词典**

评测：词典选用

评测与**词典粒度**

- 粒度较粗：收录的词较长，如“圆满完成”
  - 算法与评测应该使用**相同词典**

第二届国际中文分词评测 **SIGHAN05**

- **MSR** 语料集、词典

评测：词典未收录

评测文本中含词典未收录词条是NLP难题

- **未收录词 Out Of Vocabulary (OOV)**：俗称“新词”
  - 词典分词算法的召回率非常低
- **收录词 In Vocabulary (IV)**
  - 词典分词算法不能保证100%召回

算法	P	R	F_1	R_{OOV}	R_{IV}
最长匹配	91.80	95.69	93.71	2.58	98.22

实验：评测实现

字典树的应用

停用词过滤

停用词：不影响解决NLP任务的词

- 无实意：助词“的”，连词“和”，副词“甚至”，语气词“啊”
- 非法：敏感词，限制级

停用词过滤

停用词：不影响解决NLP任务的词

- 无实意：助词“的”，连词“和”，副词“甚至”，语气词“啊”
- 非法：敏感词，限制级

停用词过滤是一个**预处理**过程

1. 构建停用词字典
2. 替换字典中的词

实验：停用词过滤

繁简转换

朴素繁简**对照表**：繁简分歧问题

- 一简对多繁、一繁对多简

```
发现一根白头发
發現一根白頭髮
```

繁简转换

朴素繁简**对照表**：繁简分歧问题

- 一简对多繁、一繁对多简

发现一根白头发
發現一根白頭髮

- 港澳台地区习惯

繁简转换

朴素繁简**对照表**：繁简分歧问题

- 一简对多繁、一繁对多简

```
发现一根白头发
發現一根白頭髮
```

- 港澳台地区习惯

常见分类：**简体 s、繁體 t、香港繁體 hk、台灣正體 tw**

- 总计：$P(4,2)=12$种相互转换

繁简转换

朴素繁简**对照表**：繁简分歧问题

- 一简对多繁、一繁对多简

```
发现一根白头发
發現一根白頭髮
```

- 港澳台地区习惯

常见分类：**简体 s、繁體 t、香港繁體 hk、台灣正體 tw**

- 总计：$P(4,2) = 12$种相互转换
- 实际上只需要4种转换词典

实验：繁简转换

拼音转换

这里只讨论**汉字转成拼音**（拼音转汉字是输入法的研究内容）

- 多音字：需要提取词义
  - 简单方法：根据拼音词典按词转换

拼音转换

这里只讨论**汉字转成拼音**（拼音转汉字是输入法的研究内容）

- 多音字：需要提取词义
  - 简单方法：根据拼音词典按词转换

原文	重	载	不	是	重	任	！
数字音调	chong2	zai3	bu2	shi4	zhong4	ren4	none5
符号音调	chóng	zǎi	bú	shì	zhòng	rèn	none
无音调	chong	zai	bu	shi	zhong	ren	none
声调	2	3	2	4	4	4	5
声母	ch	z	b	sh	zh	r	none
韵母	ong	ai	u	i	ong	en	none
输入法头	ch	z	b	sh	zh	r	none

实验：拼音转换

词典分词总结

- 主要数据结构：字典树
- 应用：停用词过滤、繁简转换、拼音转换
- 问题：正确率不高，无法区分歧义，无法召回新词