

1.pipeline

分词：中文/英文

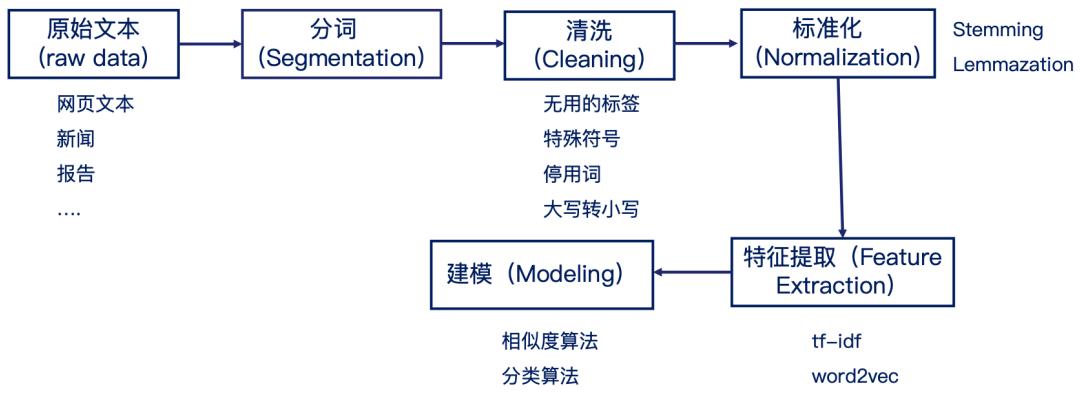
清洗：

停用词，根据场景不同选择也不同；有很多停用词的词库

标准化：对英文来说很重要

把多个单词合并成一个单词，apple&apples—》apple

需要一定的工程能力



2.word segmentation 分词

分词工具：

Jieba 分词 <https://github.com/fxsjy/jieba>

SnowNLP <https://github.com/isnowfy/snownlp>

LTP <http://www.ltp-cloud.com/>

HanNLP <https://github.com/hankcs/HanLP/>

```
> M4
# encoding=utf-8
import jieba

# 基于jieba的分词
seg_list = jieba.cut("贪心学院专注于人工智能教育", cut_all=False)
print("Default Mode: " + "/ ".join(seg_list))

jieba.add_word("贪心学院")
seg_list = jieba.cut("贪心学院专注于人工智能教育", cut_all=False)
print("Default Mode: " + "/ ".join(seg_list))
```

```
Building prefix dict from the default dictionary ...
Dumping model to file cache /var/folders/pr/0fhkrt7s4cj8yygh6m87_1fw0000gn/T/jieba.cache
Loading model cost 0.905 seconds.
Prefix dict has been built successfully.
Default Mode: 贪心/ 学院/ 专注/ 于/ 人工智能/ 教育
Default Mode: 贪心学院/ 专注/ 于/ 人工智能/ 教育
```

见 jupyter3

2.1 Max Matching 最大匹配（贪心算法）

必定有一个词典库，进行参考。

<https://blog.csdn.net/selinda001/article/details/79345072>

1) forward-max matching 前向最大匹配

max_len 参数（例如=5）

“我们经常有意见分歧”

Step1: 我们经常有一》词典库没有

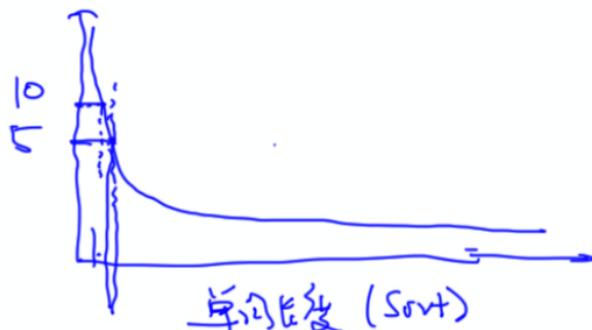
Step2: 我们经常--》词典库没有

Step3:

Step4: 我们--》词典库有

| | | | | |
|-------|-------------|---|---------|---|
| ① | [我们经常有意见分歧] | ✗ | [经常有] | ✗ |
| 0(步进) | [我们经常] | ✗ | [经常] | ✓ |
| | [我们经] | ✗ | [有意见分歧] | ✗ |
| | [我们] | ✓ | [有意见] | ✗ |
| ② | [经常有意见] | ✗ | [有意见] | ✓ |
| | [经常有意] | ✗ | ④ [分歧] | ✓ |

max_len 如何选取，越高时间复杂度越高。所以，首先参考词典库，大概百分比。



2) backward-max matching 后向最大匹配

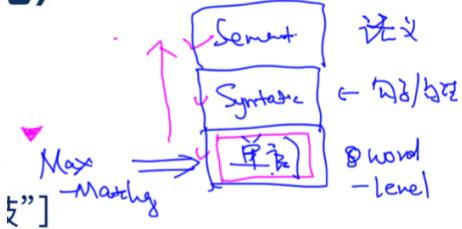
| | | | | | |
|---------|---|-----------|---|------|---|
| ① 有意见分歧 | ✗ | ② 有 经常有意见 | ✗ | ④ 我们 | ✓ |
| 意见分歧 | ✗ | 常有意见 | ✗ | | |
| 分歧 | ✗ | 有意见 | ✗ | | |
| 分歧 | ✓ | | | | |
| | | ③ 我们经常 | ✗ | | |
| | | 们经常 | ✗ | | |
| | | 经常 | ✓ | | |

前向与后向，结果基本一样。

3) 缺点

- 细分，有可能是更好
- 局部最优
- 效率低
- 歧义，不考虑语义

2) 语义的标记



2.2 Incorporate Semantic 考虑语义

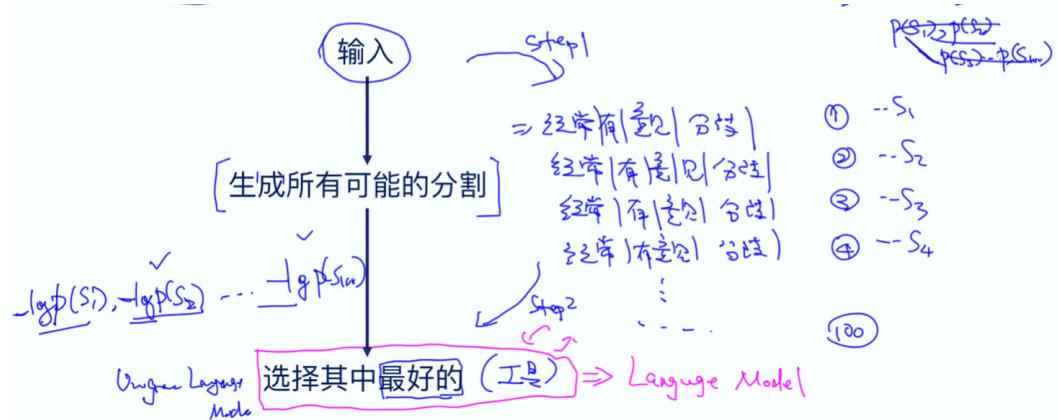
存在一个“工具”，会返回一个数值，帮助判断语义正确性。

- 例子：经常有意见分歧

词典：[“有”，“有意见”，“意见”，“分歧”，“见”，“意”，]



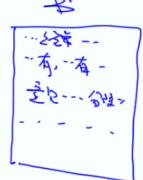
方法：输入---生成所有可能的分割---选择其中最好的（工具 language model）



Language model :

$S_1 = \text{① 经常} | \text{有} | \text{意见} | \text{分歧}$
 $S_2 = \text{② 经常} | \text{有意见} | \text{分歧}$

Unigram Model \leftarrow
 Unigram Language Model
 $P(\text{经常}) = \frac{100}{\# \text{of words}} \leftarrow$



经常: 100
 有: 1000
 意见: 500
 分歧: 200
 :

工具 \Rightarrow 语言模型 (Language Model)

$P(S_1) = 0.3$ $P(\text{经常}, \text{有}, \text{意见}, \text{分歧}) = P(\text{经常}) \cdot P(\text{有}) \cdot P(\text{意见}) \cdot P(\text{分歧}) = 0.3$
 $P(S_2) = 0.3^4$ $P(\text{经常}, \text{有意见}, \text{分歧}) = P(\text{经常}) \cdot P(\text{有意见}) \cdot P(\text{分歧}) = 0.3^3$

$$\begin{aligned}
 P(\text{我们今天上课}) &= P(\text{我们}, \text{今天}, \text{上课}) \\
 &= P(\text{我们}) P(\text{今天}) P(\text{上课})
 \end{aligned}$$

Input: 经常有意见分歧
 $\left\{ \begin{array}{l} S_1 = \text{~~~} \\ S_2 = [] \text{ } | \text{ } \text{ } \text{ } \\ S_3 = \text{~~~} \\ \vdots \\ S_{100} \end{array} \right.$

$P(\text{经常}, \text{有意见}, \text{分歧}) = P(\text{经常})P(\text{有})P(\text{意见})P(\text{分歧})$
 $= 0.0001 \cdot 0.000035 \cdot 0.000002 \cdot 0.0001 \xrightarrow{\text{溢出}} \frac{500}{10^4}$
 $= \infty \text{ or } \text{underflow}$
 计算: double / float

$P(S_1), P(S_2) \dots P(S_{100})$ $\log P(\text{经常}, \text{有意见}, \text{分歧}) = \log P(\text{经常}) + \log P(\text{有}) + \log P(\text{意见}) + \log P(\text{分歧})$
 $\log(x \cdot y) = \log x + \log y$

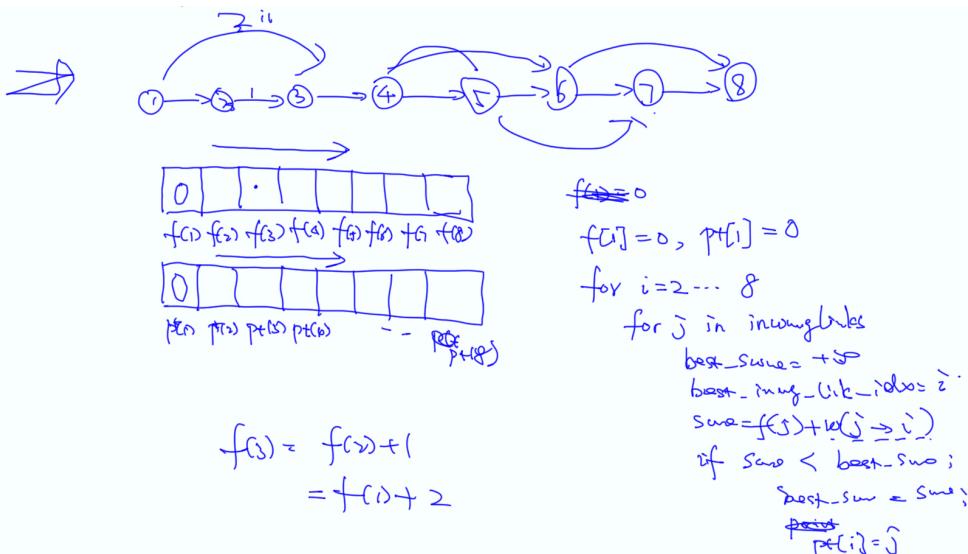
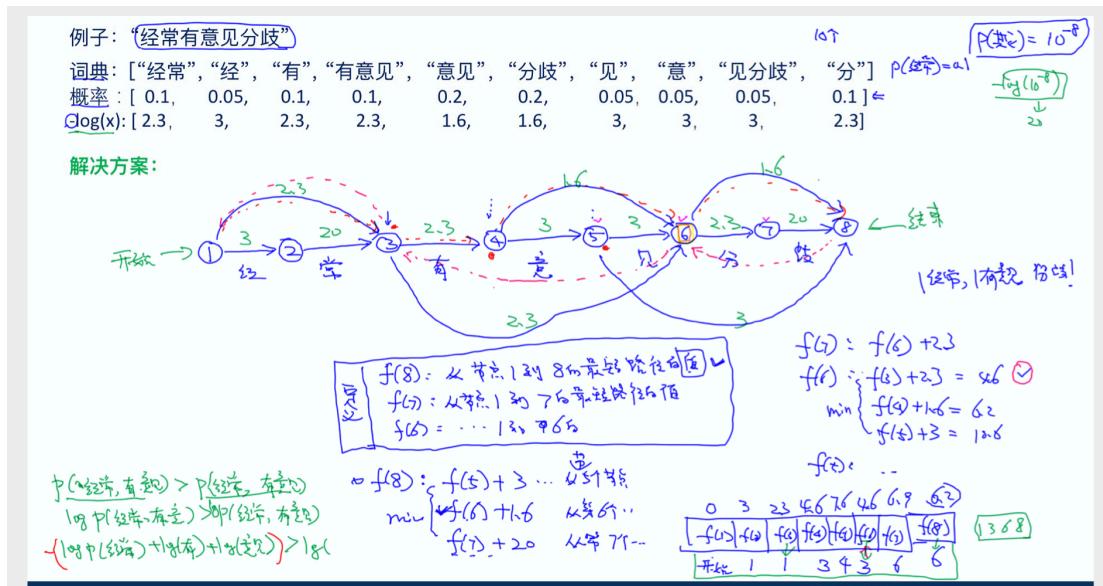
等同 $(P(S_1) > P(S_2) > P(S_3) > P(S_4))$
 $(\log P(S_1) > \log P(S_2) > \log P(S_3) > \log P(S_4))$

2.3 维特比算法

从一个、到两个字组合、到三个字组合，如此循环

目标：找路径之和最小的路径。

实质：最短路径算法



2.4 summary

知识点总结：

- 基于匹配规则的方法
- 基于概率统计方法 (LM, HMM, CRF..)
- 分词可以认为是已经解决的问题

需要掌握什么？

- 可以自行实现基于最大匹配和Unigram LM的方法

3. Spell Correction 拼写错误纠正

两类：错别字 or 语法错误（上下文）

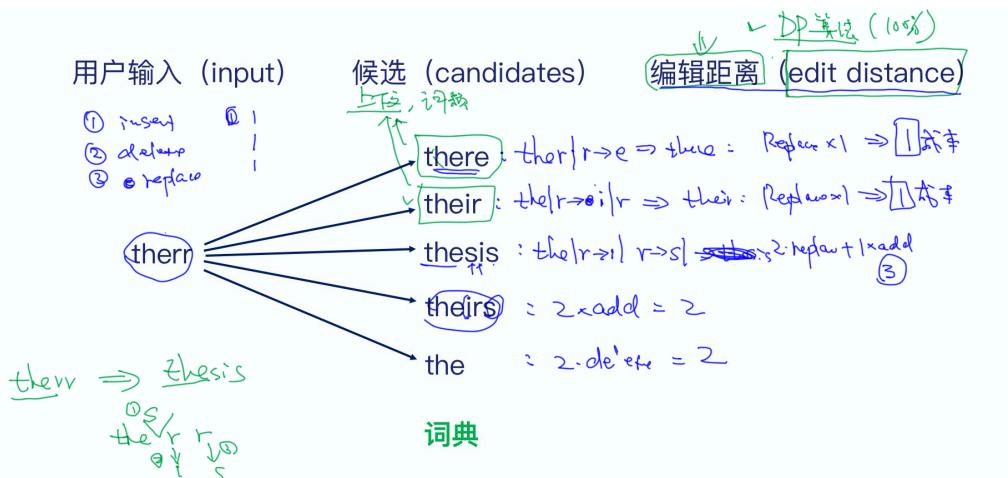
错别字：字典里不存在

3.1 edit distance

用户输入 (input) 候选 (candidates) 编辑距离 (edit distance)



Edit : insert、delete、replace



3.2 dp 算法

$$\begin{aligned} \text{DP 算法核心: } & \quad \text{insert del} \\ \text{Big Problem} \Rightarrow \text{Smaller Problem} & \quad \leftarrow \end{aligned}$$

$$S_1: \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & a & p & p & | & e & p & | & x \\ \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline \end{array} \quad S_2: \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & a & p & p & p & | & l & e & p & p \\ \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline \end{array}$$

$$d(S_1[0:6], S_2[0:8]) = d(S_1[0:5], S_2[0:7])$$

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{pmatrix} & \\ & \end{pmatrix} \begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix}$$

$$A = \begin{pmatrix} & \\ & \end{pmatrix}$$

$$A^n = (\sum D)(\sum D)$$

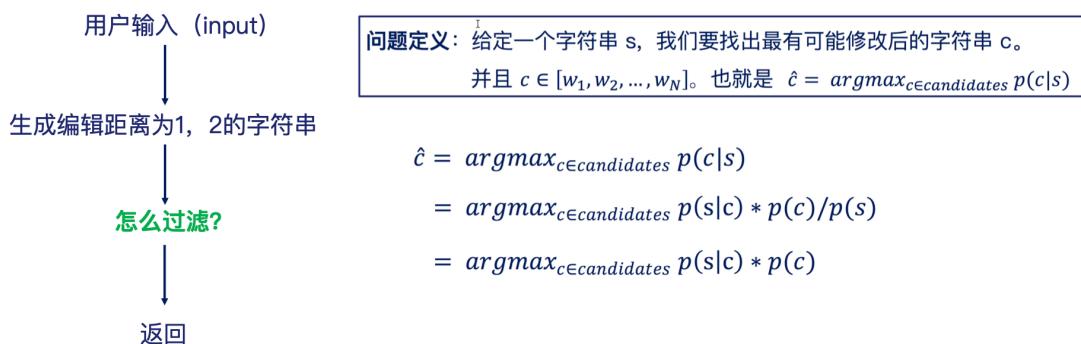
$$[]^n =$$

3.3 better way



为什么只生成 1, 2 距离：绝大部分情况正确单词都落在距离为 2 的字符串内。

Select :



S : 输错字符。C : 原本想表达的意思。

4. Words Filtering

对于 NLP 的应用，我们通常先把停用词、出现频率很低的词汇过滤掉
在英文里，比如 “the” , “an” , “their” 这些都可以作为停用词来处理。但是，也需要考虑自己的应用场景

5. Words Normalization

5.1 Stemming

went, go, going

fly, flies,

意思都类似，怎么合并？

deny, denied, denying

fast, faster, fastest

Stemming 未必把一个单词还原到词典库的词根，它有自己的规则。

Porter Stemmer 的一些规则：

<https://tartarus.org/martin/PorterStemmer/java.txt>

Step 1a

| | | | |
|------|------|----------|----------|
| sses | → ss | caresses | → caress |
| ies | → i | ponies | → poni |
| ss | → ss | caress | → caress |
| s | → Ø | cats | → cat |

Step 1b

| | | | |
|----------|-----|-----------|-----------|
| (*v*)ing | → Ø | walking | → walk |
| | | sing | → sing |
| (*v*)ed | → Ø | plastered | → plaster |

Step 2 (for long stems)

| | | | |
|---------|-------|------------|------------|
| ational | → ate | relational | → relate |
| izer | → ize | digitizer | → digitize |
| ator | → ate | operator | → operate |
| ... | | | |

Step 3 (for longer stems)

| | | | |
|------|-----|------------|----------|
| al | → Ø | revival | → reviv |
| able | → Ø | adjustable | → adjust |
| ate | → Ø | activate | → activ |
| ... | | | |

<http://facweb.cs.depaul.edu/mobasher/classes/csc575/papers/porter-algorithm.html>