分词：

切分歧义：

组合型歧义：分词粒度不同导致的不同切分结果。在文本分类，情感分析等文本分析场景下，粗粒度划分较好。而在搜索引擎场景下，为了保证recall，细粒度的划分则较好

交集型歧义：不同切分结果共用相同的字，前后组合的不同导致不同的切分结果

真歧义：本身语法或语义没有问题，即使人工切分也会产生歧义。

未登录词：

定义：也叫新词发现，或者生词，未被词典收录的词。**未登录词对于分词精度的影响远远超过歧义切分。未登录词识别难度也很大。**

中文分词算法：

分词算法主要分为两类，**基于词典的规则匹配方法**，和**基于统计的机器学习方法。**

基于词典的分词算法：

基于词典的分词算法，本质上就是**字符串匹配。**将待匹配的字符串基于一定的算法策略，和一个足够大的词典进行字符串匹配，如果匹配命中，则可以分词。根据不同的匹配策略，又分为**正向最大匹配法，逆向最大匹配法，双向匹配分词，全切分路径选择**等。

最大匹配法主要分为三种：

1. **正向最大匹配法**，从左到右对语句进行匹配，匹配的词越长越好。比如“商务处女干事”，划分为“商务处/女干事”，而不是“商务/处女/干事”。这种方式切分会有歧义问题出现，比如“结婚和尚未结婚的同事”，会被划分为“结婚/和尚/未/结婚/的/同事”。
2. **逆向最大匹配法**，从右到左对语句进行匹配，同样也是匹配的词越长越好。比如“他从东经过我家”，划分为“他/从/东/经过/我家”。这种方式同样也会有歧义问题，比如“他们昨日本应该回来”，会被划分为“他们/昨/日本/应该/回来”。
3. **双向匹配分词**，则同时采用正向最大匹配和逆向最大匹配，选择二者分词结果中**词数较少者。**但这种方式同样会产生歧义问题，比如“他将来上海”，会被划分为“他/将来/上海”。由此可见，词数少也不一定划分就正确。

全切分路径选择，将所有可能的切分结果全部列出来，从中选择最佳的切分路径。分为两种选择方法

1. **n最短路径方法。**将所有的切分结果组成有向无环图，切词结果作为节点，词和词之间的边赋予权重，找到权重和最小的路径即为最终结果。比如可以通过词频作为权重，找到一条总词频最大的路径即可认为是最佳路径。
2. **n元语法模型。**同样采用n最短路径，只不过路径构成时会考虑词的上下文关系。一元表示考虑词的前后一个词，二元则表示考虑词的前后两个词。然后根据语料库的统计结果，找到概率最大的路径。

基于统计的分词算法：

基于统计的分词算法，本质上是一个序列标注问题。将语句中的字，**按照他们在词中的位置进行标注。**标注主要有：B（词开始的一个字），E（词最后一个字），M（词中间的字，可能多个），S（一个字表示的词）。例如“网商银行是蚂蚁金服微贷事业部的最重要产品”，标注后结果为“BMMESBMMEBMMMESBMEBE”，对应的分词结果为“网商银行/是/蚂蚁金服/微贷事业部/的/最重要/产

品”。

基于统计分析方法，得到序列标注结果，就可以得到分词结果了。这类算法基于机器学习或者现在火热的深度学习，主要有HMM，CRF，SVM，以及深度学习等。

1. **HMM，隐马尔科夫模型。**隐马尔科夫模型在机器学习中应用十分广泛，它包含观测序列和隐藏序列两部分。对应到NLP中，语句是观测序列，而序列标注结果是隐藏序列。任何一个HMM都可以由一个五元组来描述：观测序列，隐藏序列，隐藏态起始概率，隐藏态之间转换概率（转移概率），隐藏态表现为观测值的概率（发射概率）。其中起始概率，转移概率和发射概率可以通过大规模语料统计来得到。从隐藏态初始状态出发，计算下一个隐藏态的概率，并依次计算后面所有的隐藏态转移概率。序列标注问题就转化为了求解概率最大的隐藏状态序列问题。标签集合：{"S", "B", "M", "E"}

   初始概率 π定义：

   $$\pi_i = P(z_1 = s_i)$$

   统计方式：统计每句话第一个字的标签，归一化。

   示例：

   - 第一个字是 S 的有 2 次
   - 第一个是 B 的有 1 次
   - 所以：

   $$\pi_S = \frac{2}{3}, \quad \pi_B = \frac{1}{3}$$

   转移概率 A定义：

   $$A_{ij} = P(z_t = s_j \mid z_{t-1} = s_i)$$

   统计方式：统计相邻标签对的出现次数，归一化。

   示例：

   - S→S 出现 1 次
   - S→B 出现 2 次
   - 所以：

   $$A_{S\to S} = \frac{1}{3}, \quad A_{S\to B} = \frac{2}{3}$$

   发射概率 B定义：

   $$B_{ik} = P(x_t = o_k \mid z_t = s_i)$$

   统计方式：统计每个状态下输出的字符频率，归一化。

   示例：

   - 状态 S 下，“我”出现 1 次，S 总共出现 4 次 →

   $$B_{S,\text{我}} = \frac{1}{4}$$

   - 状态 B 下，“北”出现 1 次，B 总共出现 3 次 →

   $$B_{B,\text{北}} = \frac{1}{3}$$

（**jieba分词中使用HMM模型来处理未登录词**问题，并利用viterbi算法来计算观测序列（语句）最可能的隐藏序列（BEMS标注序列）。

2. **CRF，条件随机场。**也可以描述输入序列和输出序列之间关系。只不过它是基于条件概率来描述模型的。详细的这儿就不展开了。
3. **深度学习。**将语句作为输入，分词结果作为标注，可以进行有监督学习。训练生成模型，从而对未知语句进行预测。

词性标注：

词性标注中的难点主要有

1. 相对于英文，中文缺少词形态变化，**不能从词的形态来识别词性**
2. **一词多词性很常见。**
3. **词性划分标准不统一。**词类划分粒度和标记符号等，目前还没有一个广泛认可的统一的标准。
4. **未登录词问题。**和分词一样，未登录词的词性也是一个比较大的课题。

词性标注算法：

词性标注算法也分为两大类，**基于字符串匹配的字典查找算法**和**基于统计的算法**

基于字符串匹配的字典查找算法：

先对语句进行分词，然后从字典中查找每个词语的词性，对其进行标注即可。
（在jieba分词中的词典的一部分词语。每一行对应一个词语，分为三部分，分别为**词语名 词数 词性。**因此分词完成后只需要在字典中查找该词语的词性即可对其完成标注。）

基于统计的词性标注算法：

也可以**通过HMM隐马尔科夫模型来进行词性标注。**观测序列即为分词后的语句，隐藏序列即为经过标注后的词性标注序列。起始概率 发射概率和转移概率和分词中的含义大同小异，可以通过大规模语料统计得到。观测序列到隐藏序列的计算可以通过viterbi算法，利用统计得到的起始概率 发射概率和转移概率来得到。得到隐藏序列后，就完成了词性标注过程。

jieba词性标注原理：

jieba在分词的同时，可以进行词性标注。利用`jieba.posseg`模块来进行词性标注，会给出分词后每个词的词性。
对`pseg.cut()`进行详细的分析，其主要流程为

1. **准备工作**：check字典是否初始化好，如果没有则先初始化字典。将语句转为UTF-8或者GBK。根据正则匹配，将输入文本分隔成一个个语句。
2. **遍历语句list，对每个语句进行单独分词和词性标注。**
3. **对于未登录词，使用HMM隐马尔科夫模型处理**

句法分析：

句法分析也是自然语言处理中的基础性工作，它分析句子**的句法结构（主谓宾结构）和词汇间的依存关系（并列，从属等）。**通过句法分析，可以为语义分析，情感倾向，观点抽取等NLP应用场景打下坚实的基础。句法结构分析，识别句子的主谓宾、定状补，**并分析各成分之间的关系**.

句法结构分析：

通过句法结构分析，就能够分析出语句的主干，以及各成分间关系。对于复杂语句，仅仅通过词性分析，不能得到正确的语句成分关系。句法结构分析的标注如下：

关系类型	Tag	Description	Example
主谓关系	SBV	subject-verb	我送她一束花 (我 <-- 送)
动宾关系	VOB	直接宾语，verb-object	我送她一束花 (送 --> 花)
间宾关系	IOB	间接宾语，indirect-object	我送她一束花 (送 --> 她)
前置宾语	FOB	前置宾语，fronting-object	他什么书都读 (书 <-- 读)
兼语	DBL	double	他请我吃饭 (请 --> 我)
定中关系	ATT	attribute	红苹果 (红 <-- 苹果)
状中结构	ADV	adverbial	非常美丽 (非常 <-- 美丽)
动补结构	CMP	complement	做完了作业 (做 --> 完)
并列关系	COO	coordinate	大山和大海 (大山 --> 大海)
介宾关系	POB	preposition-object	在贸易区内 (在 --> 内)
左附加关系	LAD	left adjunct	大山和大海 (和 <-- 大海)
右附加关系	RAD	right adjunct	孩子们 (孩子 --> 们)
独立结构	IS	independent structure	两个单句在结构上彼此独立
标点	WP	punctuation	。
核心关系	HED	head	指整个句子的核心

语义依存关系分析：

识别词汇间的从属、并列、递进等关系，可以获得较深层的语义信息。

句法分析工具：

哈工大LTP：[语言云（语言技术平台云 LTP-Cloud）]
斯坦福句法分析工具Stanford Parser：[The Stanford Natural Language Processing Group]

深度学习方法：

研究表明，很多情况下，单纯的bi-lstm（双向LSTM），比基于句法分析树的tree-lstm效果更好

	Bi-LSTM	Tree-LSTM
细粒度情感分类	49.8 / 50.7（以标点符号分割）	50.4
二元情感分类	79.0	77.4
问答对匹配	56.4	55.8
语义关系分类	**75.2**	**76.7**
Discourse 分析	57.5	56.4

这主要是因为当前句法分析准确度不高，只有90%左右。如果是句子成分关系很复杂，则准确率更低。因此给lstm网络带来了很大的噪声，从而导致了tree-lstm模型准确度的降低。**但是tree-lstm可以使用较少的标注语料，而且在句子结构复杂的长语句上，表现更好。因此当语料较少且句子结构很复杂时，可以考虑使用tree-lstm。**

词向量：

词向量一方面解决了词语的编码问题，另一方面也解决了词的同义关系

工具：word2vec

它可以进行词向量训练，加载已有模型进行增量训练，求两个词向量相似度，求与某个词接近的词语，等等。词向量模型训练只需要有训练语料即可，语料越丰富准确率越高，属于**无监督学习。**

模型训练：

```
# gensim是自然语言处理的一个重要Python库，它包括了Word2vec
import gensim
from gensim.models import word2vec

# 语句，由原始语句经过分词后划分为的一个个词语
sentences = [['网商银行', '体验', '好'], ['网商银行','转账','快']]

# 使用word2vec进行训练
# min_count: 词语频度，低于这个阈值的词语不做词向量
# size:每个词对应向量的维度，也就是向量长度
# workers：并行训练任务数
model = word2vec.Word2Vec(sentences, size=256, min_count=1)

# 保存词向量模型，下次只需要load就可以用了
model.save("word2vec_atec")
```

增量训练：

有时候我们语料不是很丰富，但都是针对的某个垂直场景的，比如网商银行相关的语料。此时训练词向量时，可以先基于一个已有的模型进行增量训练，这样就可以得到包含特定语料的比较准确的词向量了。

```
# 先加载已有模型
model = gensim.models.Word2Vec.load("word2vec_atec")

# 进行增量训练
corpus = [['网商银行','余利宝','收益','高'],['贷款','发放','快']] # 新增语料
model.build_vocab(corpus, update=True)  # 训练该行
model.train(corpus, total_examples=model.corpus_count, epochs=model.iter)

# 保存增量训练后的新模型
model.save("../data/word2vec_atec")
```

词语相似度：

```
# 验证词相似程度
print model.wv.similarity('花呗'.decode('utf-8'), '借呗'.decode('utf-8'))
```

词语相近的多个词语：

```
for i in model.most_similar(u"我"):
    print i[0],i[1]
```

词训练算法：

词向量可以通过使用大规模语料进行无监督学习训练得到，常用的算法有`CBOW`连续词袋模型和`skip-gram`跳字模型。二者没有本质的区别，算法框架完全相同。区别在于，**CBOW利用上下文来预测中心词。而skip-gram则相反，利用中心词来预测上下文。**比如对于语料`{“The”, “cat”, “jump”, “over”, “the”, “puddle”}`，CBOW利用上下文`{“The”, “cat”, “over”, “the”, “puddle”}`预测中心词“jump”，而skip-gram则利用jump来预测上下文的词，比如jump->cat, jump->over。一般来说，**CBOW适合小规模训练语料，对其进行平滑处理。skip-gram适合大规模训练语料，可以基于滑窗随机选择上下文词语。**word2vec模型训练时默认采用skip-gram。

词向量算法实现：

流程还是很简单的，关键在第四步batch的构建，和第五步训练模型的构建，步骤如下

1. 下载语料文件，并校验文件字节数是否正确。想得到比较准确的词向量，一般需要通过爬虫获取维基百科，网易新闻等既丰富又相对准确的语料素材。一般需要几十上百G的corpus，即语料。谷歌根据不同的语料预训练了一些词向量，参考 Embedding/Chinese-Word-Vectors
2. 语料处理，文本切割为一个个词语。英文的话以空格为分隔符进行切分即可（有误差，但还好）。中文的话需要通过分词工具进行分割。
3. 词表制作，词语预编码。根据词语出现频率排序，序号代表这个单词。词语编码的一种常用方式。
4. 生成训练的batch label对。这是比较关键的一步，也是体现skip-gram算法的一步。

- 先取出滑窗范围的一组词，如滑窗大小为5，则取出5个词。
- 位于中心的词为中心词，比如滑窗大小为5，则第三个词为中心词。其他词则称为上下文。
- 从上下文中随机取出`num_skip`个词，比如`num_skip`为2，则从4个上下文词语中取2个。通过随机选取提高了一定的泛化性
- 得到`num_skip`个中心词->上下文的x->y词组
- 将滑窗向右移动一个位置，继续这些步骤，直到滑窗到达文本最后

1. **构造训练模型**，这一步也很关键。利用nce loss将多分类问题转化为二分类问题，optimizer优化方法采用随机梯度下降。

   NCE（Noise Contrastive Estimation）的解决方案：**不直接预测所有词的概率分布而是将问题转化为二分类问题**：

   - 判断一个词是否是当前中心词的上下文词（正例）
   - 判断其他词是否是当前中心词的上下文词（负例）
2. 开始真正的训练。这一步比较常规化。送入第四步构建的batch进行feed，跑optimizer和loss，并进行相关信息打印即可。训练结束后，即可得到调整完的词向量模型

关键参数：

参数	含义	示例
`skip_window`	上下文窗口大小（左右各取多少个词）	`skip_window=1` → 左右各取1个词
`num_skips`	每个中心词对应多少个上下文词	`num_skips=2` → 每个中心词生成2个样本
`batch_size`	每个 batch 包含多少个样本	`batch_size=8` → 每次训练8个样本

示例：模拟生成 batch-label 对

假设句子：

["我", "爱", "自然语言", "处理"]

设置参数：

- `skip_window = 1` → 窗口大小为 3（中心词 + 左右各1个词）
- `num_skips = 2` → 每个中心词生成 2 个样本
- `batch_size = 4` → 每个 batch 包含 4 个样本

步骤 1：滑动窗口遍历句子

我们用一个滑动窗口遍历整个句子：

步骤	窗口内容	中心词	上下文词
1	["我", "爱", "自然语言"]	"爱"	["我", "自然语言"]
2	["爱", "自然语言", "处理"]	"自然语言"	["爱", "处理"]

步骤 2：从上下文中随机选取 `num_skips` 个词

对于每个中心词，从上下文中随机选取 `num_skips` 个词：

第一步：中心词 "爱"

- 上下文词：["我", "自然语言"]
- 随机选取 2 个（num_skips=2）→ 得到：
  - ("爱", "我")
  - ("爱", "自然语言")

第二步：中心词 "自然语言"

- 上下文词：["爱", "处理"]
- 随机选取 2 个（num_skips=2）→ 得到：
  - ("自然语言", "爱")

- ("自然语言", "处理")

步骤 3：构建 batch 和 labels

将上面的样本整理为 batch 和 labels：

batch（中心词）	label（上下文词）
"爱"	"我"
"爱"	"自然语言"
"自然语言"	"爱"
"自然语言"	"处理"

步骤 4：编码为数字编号

假设词表如下：

词	编号
我	1
爱	2
自然语言	3
处理	4

则 batch 和 labels 变为：

batch = [2, 2, 3, 3] # "爱" 和 "自然语言" 的编号
labels = [1, 3, 2, 4] # "我"、"自然语言"、"爱"、"处理" 的编号