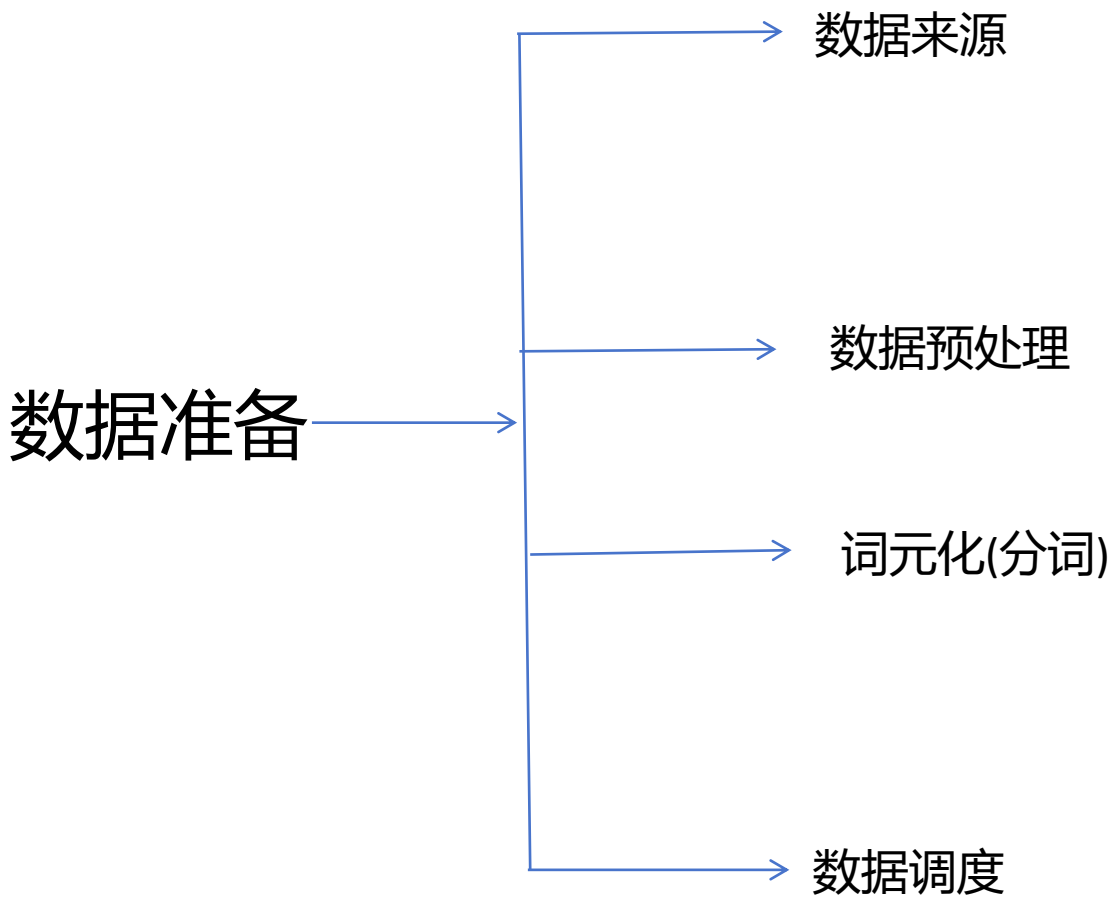
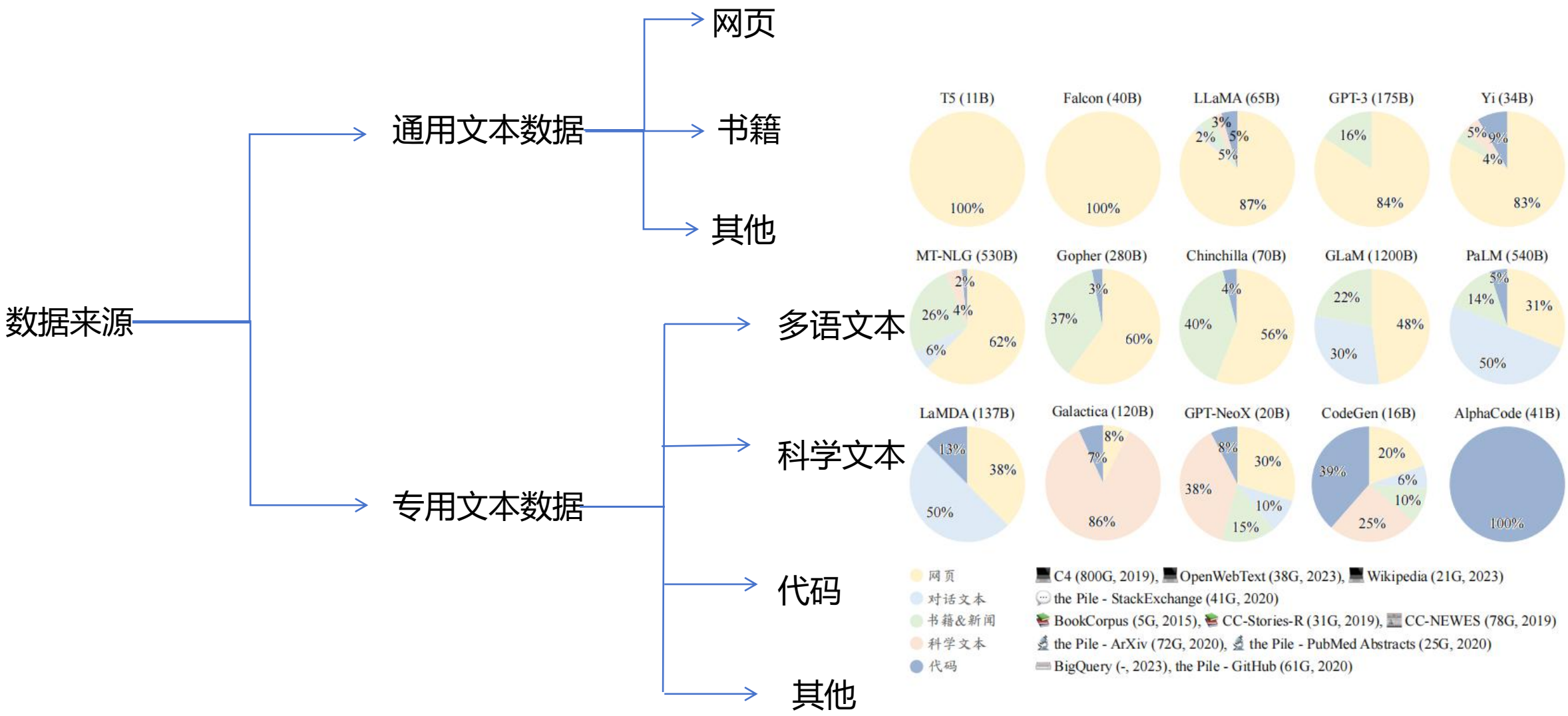
数据准备

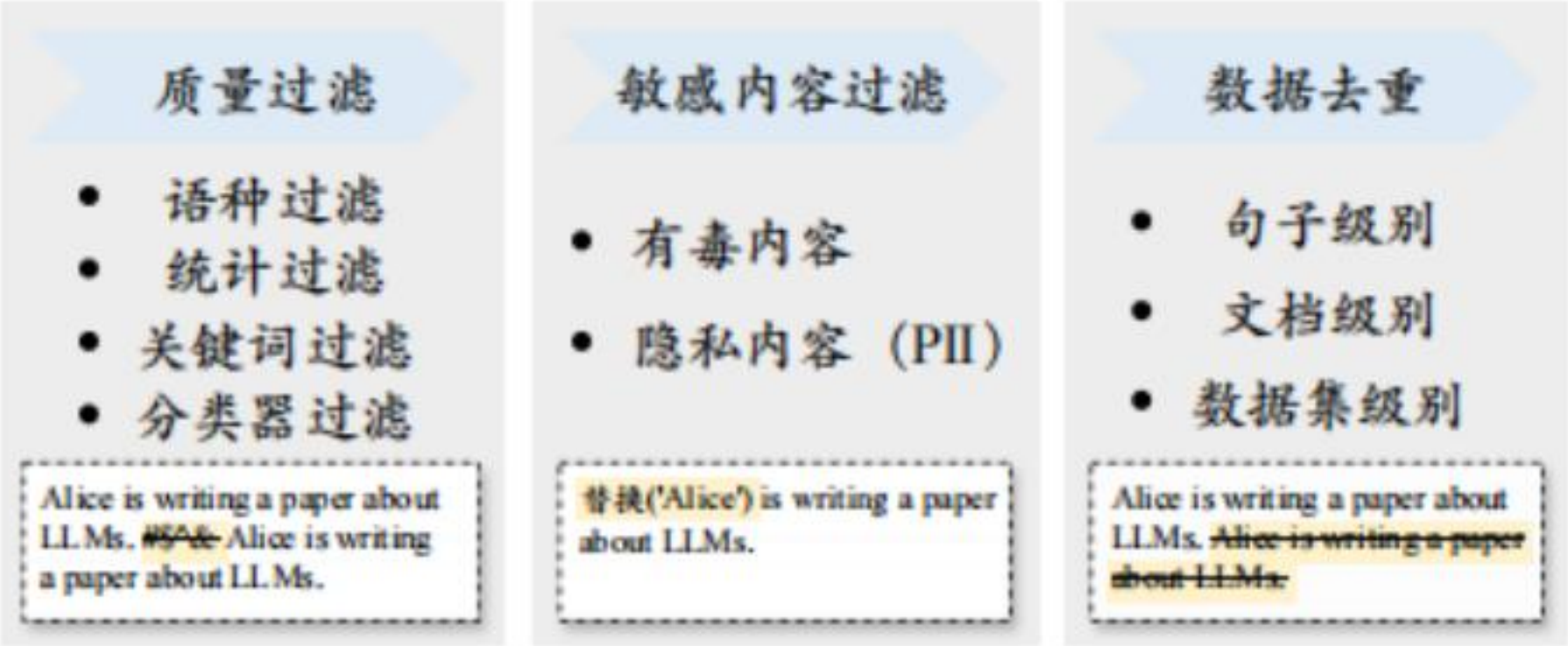
刘任强

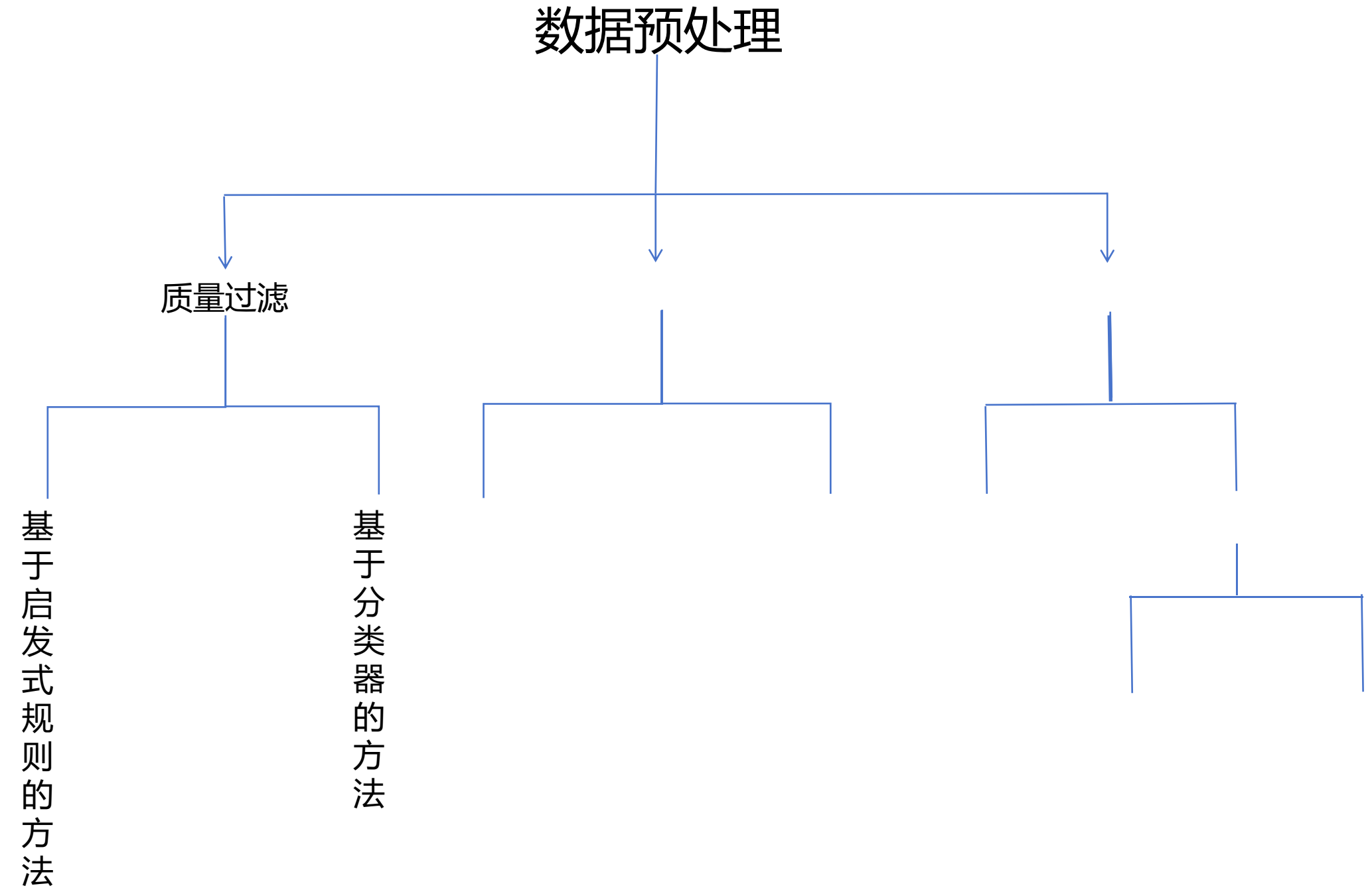

数据准备
数据来源
数据预处理
词元化(分词)
数据调度


数据来源
通用文本数据
网页
书籍
其他
专用文本数据
多语文本
科学文本
代码
其他
T5 (11B)
Falcon (40B)
LLaMA (65B)
GPT-3 (175B)
Yi (34B)
MT-NLG (530B)
Gopher (280B)
Chinchilla (70B)
GLaM (1200B)
PaLM (540B)
LaMDA (137B)
Galactica (120B)
GPT-NeoX (20B)
CodeGen (16B)
AlphaCode (41B)
网页
对话文本
书籍&新闻
科学文本
代码
C4 (800G, 2019), OpenWebText (38G, 2023), Wikipedia (21G, 2023)
the Pile - StackExchange (41G, 2020)
BookCorpus (5G, 2015), CC-Stories-R (31G, 2019), CC-NEWES (78G, 2019)
the Pile - ArXiv (72G, 2020), the Pile - PubMed Abstracts (25G, 2020)
BigQuery (-, 2023), the Pile - GitHub (61G, 2020)

数据预处理

目的：消除低质量、冗余、无关甚可能有害的数据




数据预处理
质量过滤
基于启发式规则的方法
基于分类器的方法

质量过滤

两种数据清洗的方法：

1. 基于启发式规则的方法

其常见的数据清洗方法：

基于语种的过滤：训练特定语言，过滤其他语言。(注：非英，...保英高)

基于简单统计指标的过滤：标点符号分布、句子长度(特征)

基于关键词过滤

基于困惑度(Perlexity)过滤

2. 基于分类器方法

困惑度(Perplexity)：

是衡量语言模型预测能力的一个重要指标，他反映了模型对预测数据的预测好坏程度。困惑度越低，表示模型在预测下一个词时的不确定性越小，模型的性能越好。但在实际应用中，**单一使用困惑度效果不佳**，所以需要与其他评估指标结合使用，以获得更准确的结果。

加入其他评估指标：

BLEU ：

用于机器翻译任务的评估指标，通过比较机器翻译输出与一组参考翻译之间的n-gram重叠程度来评估翻译质量。BLEU分数越高，表示翻译质量越接近人类翻译。

ROUGE ：

这个指标主要用于评估自动摘要的质量，通过计算摘要中与参考摘要共有的n-gram数量来评估摘要的准确性和完整性。最常用的是 ROUGE-N 和 ROUGE-L。

EM ：

这是一个简单的评估指标，用于检查模型生成的输出是否与参考答案完全匹配。在某些任务中，如问答系统，EM可以作为一个直接的指标来衡量模型性能。

困惑度

给定一个语言模型和一个测试序列 $w = w_1, w_2, \ldots, w_N$，困惑度 PP 的定义如下：

$$PP(w) = P(w)^{-\frac{1}{N}} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_1, w_2, \ldots, w_{i-1})\right)$$

其中：

- $P(w)$ 是生成序列 w 的概率。
- N 是序列中的单词数。
- $P(w_i|w_1, w_2, \ldots, w_{i-1})$ 是给定之前的单词序列 $w_1, w_2, \ldots, w_{i-1}$ 时，模型对当前单词 w_i 的条件概率。

困惑度

假设我们有一个语言模型，我们想计算句子 "I love natural language processing" 的困惑度：

1. 对于句子中的每个单词，使用模型计算条件概率：
   - $P(I)$
   - $P(love|I)$
   - $P(natural|I, love)$
   - $P(language|I, love, natural)$
   - $P(processing|I, love, natural, language)$
2. 计算对数概率：

$$\text{log\_prob} = \log P(I) + \log P(love|I) + \log P(natural|I, love) + \log P(language|I, love, natural) + \log P(processing|I, love, natural, language)$$

3. 计算总的单词数 $N = 5$。
4. 计算困惑度：

$$PP(w) = \exp\left(-\frac{1}{5} \cdot \text{log\_prob}\right)$$

BLEU计算

用于机器翻译任务的评估指标，通过比较机器翻译输出与一组参考翻译之间的n-gram重叠程度来评估翻译质量。BLEU分数越高，表示翻译质量越接近人类翻译。

计算步骤：

- n-gram匹配：计算候选文本与参考文本之间的n-gram匹配数。n-gram是连续的n个词的组合。
- 精确度计算：对于每个n-gram，计算候选文本与参考文本匹配的n-gram的个数。

  精确度公式为：

$$p_n = \frac{\text{匹配的 n-gram 数}}{\text{候选文本中的 n-gram 总数}}$$

- 加权平均：通常计算1-gram到4-gram的精确度，BLEU计算公式为：

$$BLEU = BP \times \exp\left(\sum_{n=1}^{N} w_n \log(p_n)\right)$$

其中：

- BP 是惩罚因子（Brevity Penalty），用于惩罚生成文本的长度，如果生成文本的长度小于参考文本的长度。
- w_n 是每种 n-gram 权重，通常是 $\frac{1}{N}$（N=4）。
- p_n 是 n-gram 精确度。

Brevity Penalty：BP公式：

$$BP = \begin{cases} 1 & \text{如果候选文本长度} > \text{参考文本长度} \\ e^{(1-\frac{r}{c})} & \text{如果候选文本长度} \leq \text{参考文本长度} \end{cases}$$

其中 r是参考文本的长度，c 是候选文本的长度

例子

现有的文本
候选文本(Cnadidate):”The cat is on the mat.”
参考文本(Reference):”The cat is sittiong on the mat.”

1.生成n-gram

1-gram:

- Candidate: {The, cat, is, on, the, mat}
- Reference: {The, cat, is, sitting, on, the, mat}

2-gram:

- Candidate: {The cat, cat is, is on, on the, the mat}
- Reference: {The cat, cat is, is sitting, sitting on, on the, the mat}

2.计算匹配

1-gram 匹配:

- 匹配的1-gram: {The, cat, is, on, the, mat} ⟶ 5个匹配
- 候选文本的1-gram总数 = 6
- $p_1 = \frac{5}{6}$

2-gram 匹配:

- 匹配的2-gram: {The cat, cat is, on the, the mat} ⟶ 4个匹配
- 候选文本的2-gram总数 = 5
- $p_2 = \frac{4}{5}$

3.计算BELU分数

选取 N = 2（可以选择到 4）

计算 BP:

- $r = 7$（参考文本长度），$c = 6$（候选文本长度），所以
- $BP = e^{(1-\frac{7}{6})} = e^{-0.1667} \approx 0.846$

综合计算:

$$BLEU = BP \times \exp\left(\frac{1}{2}\log(p_1) + \frac{1}{2}\log(p_2)\right)$$

$$BLEU \approx 0.846 \times \exp\left(\frac{1}{2}(-0.182) + \frac{1}{2}(-0.223)\right)$$

$$BLEU \approx 0.846 \times \exp(-0.2025) \approx 0.846 \times 0.817 \approx 0.692$$

应用到质量过滤中(How) : 结合 BLEU 分数和困惑度，可以为 NLP 模型生成的文本建立一个过滤系统。

指标设置：

1.设定 BLEU 分数的阈值

2.设定困惑度的阈值

生成文本的质量评估：

1.对每个生成的文本使用上述代码进行评估。

2.只保留 BLEU 分数高于阈值且困惑度低于阈值的文本，认为这文本质量较好，符合预期。

ROUGE计算

主要用于评估自动摘要的质量，通过计算摘要中与参考摘要共有的n-gram数量来评估摘要的准确性和完整性。最常用的是 ROUGE-N 和 ROUGE-L。

ROUGE-N:定义: ROUGE-N 衡量的是 n-grams（n元组）的重叠情况。通常使用 ROUGE-1（单词级基于单元）和 ROUGE-2（双词级基于单元)。

计算公式:

$$\text{Precision} = \frac{\text{Number of overlapping n-grams}}{\text{Number of n-grams in the candidate}}$$

$$\text{Recall} = \frac{\text{Number of overlapping n-grams}}{\text{Number of n-grams in the reference}}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\textbf{ROUGE-N} = \frac{\sum_{\text{gram}_n \in \text{Ref}} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{\text{gram}_n \in \text{Ref}} \text{Count}(\text{gram}_n)}$$

其中：

$\text{Count}_{\text{match}}(\text{gram}_n)$ 是生成文本中与参考文本匹配的n-gram数量。

$\text{Count}(\text{gram}_n)$ 是参考文本中的n-gram总数。

ROUGE-L:定义: ROUGE-L 衡量的是最长公共子序列（LCS）的重叠情况。

计算公式:

$$\text{Precision} = \frac{\text{Length of LCS}}{\text{Total tokens in the candidate}}$$

$$\text{Recall} = \frac{\text{Length of LCS}}{\text{Total tokens in the reference}}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\textbf{ROUGE-L} = \frac{LCS(\text{Gen}, \text{Ref})}{\textbf{Length of Reference}}$$

其中：

$LCS(\text{Gen}, \text{Ref})$ 是生成文本与参考文本之间的最长公共子序列的长度。

Length of Reference 是参考文本的总词数。

例子

计算 ROUGE-1

1. **提取 n-grams**:
   - 候选摘要的单词:
     - n-grams: ["The", "cat", "sat", "on", "the", "mat"]
   - 参考摘要的单词:
     - n-grams: ["The", "cat", "is", "sitting", "on", "the", "mat"]
2. **计算重叠 n-grams**:
   - 重叠: ["The", "cat", "on", "the", "mat"]（5 个重叠词）
3. **计算 ROUGE-1 指标**:
   - Precision:

$$\text{Precision} = \frac{5}{6} \approx 0.8333$$

- Recall:

$$\text{Recall} = \frac{5}{7} \approx 0.7143$$

- F1 Score:

$$\text{F1 Score} = 2 \times \frac{0.8333 \times 0.7143}{0.8333 + 0.7143} \approx 0.7692$$

计算 ROUGE-L

1. **计算 LCS**:
   - "The cat on the mat" 是一个最长公共子序列，LCS 的长度为 5。
2. **计算 ROUGE-L 指标**:
   - Precision:

$$\text{Precision} = \frac{5}{6} \approx 0.8333$$

- Recall:

$$\text{Recall} = \frac{5}{7} \approx 0.7143$$

- F1 Score:

$$\text{F1 Score} = 2 \times \frac{0.8333 \times 0.7143}{0.8333 + 0.7143} \approx 0.7692$$

在评估 ROUGE 指标的效果时，通过以下几个方面来得出结论：

1. 综合评估 F1 分数：

ROUGE 指标通常使用 F1 分数作为主要评估标准。高的 F1 分数表明生成文本在内容和表达上与参考文本相似度较高。

2. 分析精确率和召回率：

高精确率意味着生成的文本包含了大量正确的内容，但可能遗漏了一些重要的点。

高召回率则表示模型能覆盖较多的参考内容，但可能包含了不必要或冗余的信息。

3. 与基线进行比较：

将 ROUGE 分数与基线模型的分数进行比较，评估模型的表现。

如果模型的 ROUGE 分数显著高于基线，则说明采用的模型或方法在生成内容方面有较好的效果。

EM

简单的评估指标，用于检查模型生成的输出是否与参考答案完全匹配。在某些任务中，如问答系统，EM可以作为一个直接的指标来衡量模型性能。

EM 的计算公式可以表示为：

$$\text{EM} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \times 100\%$$

例子　假设我们有以下几个问题及其参考答案和模型生成的答案：

问题: "What is the capital of France?"
参考答案: "Paris"
模型生成的答案: "Paris"
匹配结果: ✔　（完全匹配）

问题: "What is 2 + 2?"
参考答案: "4"
模型生成的答案: "Four"
匹配结果: ✘ (不匹配)

问题: "What color is the sky?"
参考答案: "Blue"
模型生成的答案: "Blue"
匹配结果: ✔　（完全匹配）

问题: "What is the largest mammal?"
参考答案: "Blue whale"
模型生成的答案: "whale"
匹配结果: ✘ (不匹配)

- 正确的预测数量: 2（问题1和问题3）
- 总的预测数量: 4（所有问题）

根据 EM 的公式：

$$\text{EM} = \frac{2}{4} \times 100\% = 50\%$$

基于分类器方法

实现分类器的方法

- 轻量级模型（如 FastText 等）
- 可微调的预训练语言模型（如 BERT、 BART 或者 LLaMA 等）
- 闭源大语言模型 API（如GPT-4、Claude 3）。

优缺点

- 轻量级模型：效率较高，但是分类的准确率和精度可能受限于模型能力。
- 预训练语言模型：可以针对性微调，但是分类性能的通用性和泛化性仍然有一定的限制；
- 闭源大语言模型：能力较强，但是无法灵活针对任务进行适配，而且用于预训练数据清洗需要花费较高的成本。

- for后两种方法来说，除了简单地进行数据过滤，还可以针对性进行数据的改写，从而使得一些整体质量还不错、但存在局部数据问题的文本仍然可以被保留下来使用。

质量过滤具体流程

数据收集与预处理：

数据收集：从各种来源(如网页，数据库等)收集大量的原始数据。确保数据的多样性和代表性，以覆盖不同的应用场景和用户需求。

数据预处理：对数据进行清洗，去除噪声、错误和重复信息。对数据进行格式化和标准化处理，确保数据的一致性和可比性。

质量评估指标确定：

明确质量标准：根据应用场景和模型需求，确定数据质量的评估指标。

选择评估方法：采用人工评估、自动化评估或两者结合的方式对数据进行质量评估。

自动化评估可以基于规则、模型或统计方法来实现。

质量过滤算法与模型应用：

基于规则的过滤

基于模型的过滤：训练文本分类器或其他机器学习模型来判数据质量。使用训练好的模型对大量数据 进行快速、准确的过滤。

集成过滤方法以达到更好的过滤效果。

采用多层次、多阶段的过滤策略，逐步剔除低质量数据。

过滤效果评估与优化：

评估过滤效果：对过滤后的数据进行质量评估，以验证过滤方法的有效性。采用合适的评估指标来衡量过滤效果。

优化过滤方法：根据评估结果对过滤方法进行调整和优化。

改进规则设计、模型训练或集成策略，以提高过滤效果。

整合结果：

汇总和分析质量报告：根据分类和处理结果，生成质量报告，包含输出的统计信息、质量指标分布等。

优化模型的依据：基于质量报告调整模型参数或训练数据，持续改善模型输出质量。

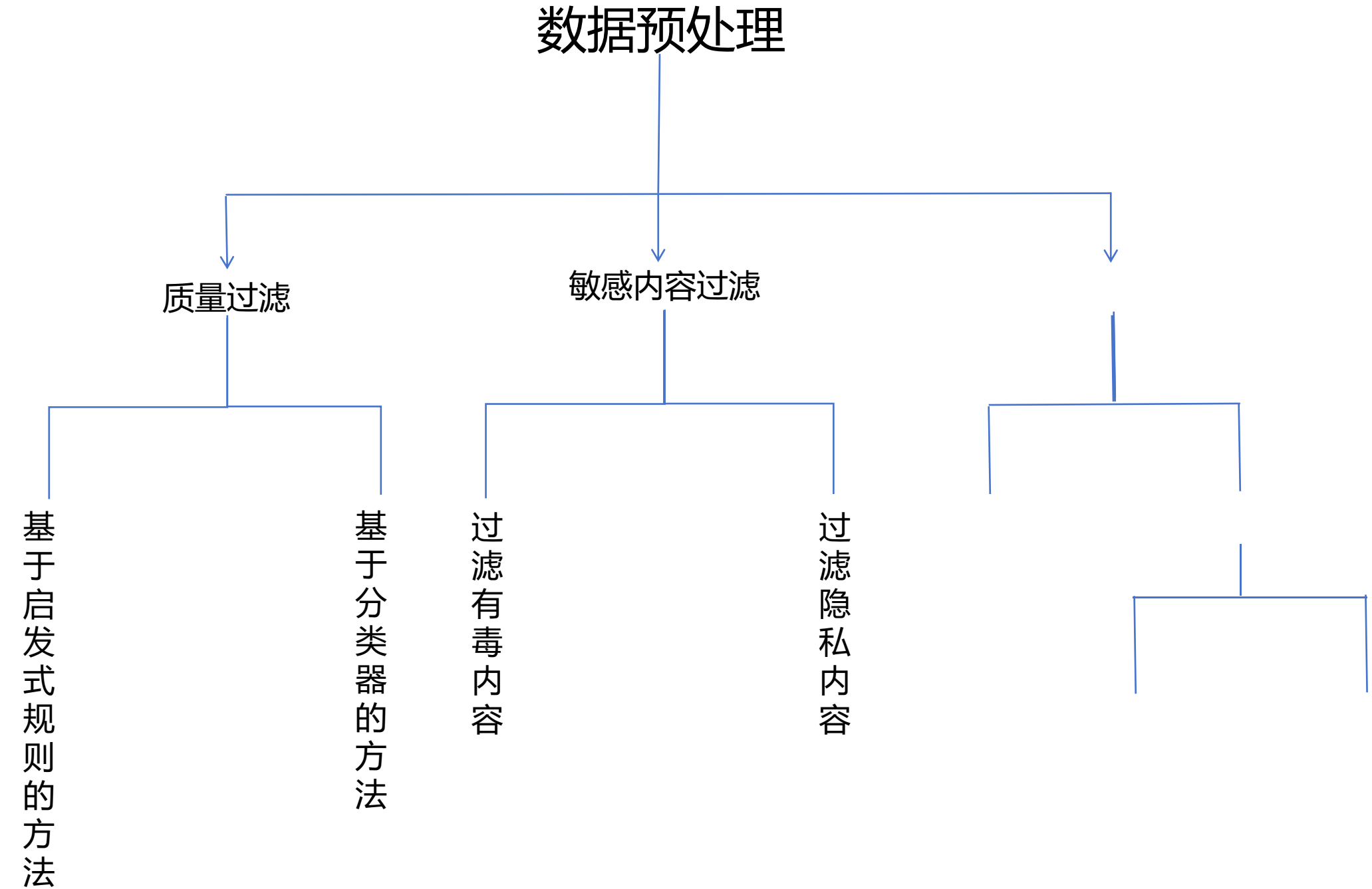
根据文本的语言过滤

```python
from utils.evaluator import LangIdentifier

class FilterPassageByLangs():
    def __init__(self) -> None:
        # 使用 LangIdentifier 模块加载已经训练好的 fasttext 模型
        self.language_identifier = LangIdentifier(model_path="utils/models/fasttext/lid.176.bin")
        self.reject_threshold = 0.5
    def filter_single_text(self, text: str, accept_lang_list: list) -> bool:
        # 使用 fasttext 模型给 text 打分，每种语言生成一个置信分数
        labels, scores = self.language_identifier.evaluate_single_text(text)
        # 如果 text 所有语言的分数均比 reject_threshold 要低，则直接定义为未知语言
        if socres[0] < self.reject_thereshold:
            labels = ["uk"]
        accept_lang_list = [each.lower() for each in accept_lang_list]
        # 如果分数最高的语言标签不在配置文件期望的语言列表中，则丢弃该文本
        if labels[0] not in accept_lang_list:
            return True
        return False
```

reject_threshold:拒绝阈值，当置信分数都低于它，标记为未知语言。

fasttext模型：词向量生成、文本分类、未登录词处理


数据预处理
质量过滤
敏感内容过滤
基于启发式规则的方法
基于分类器的方法
过滤有毒内容
过滤隐私内容

敏感内容过滤

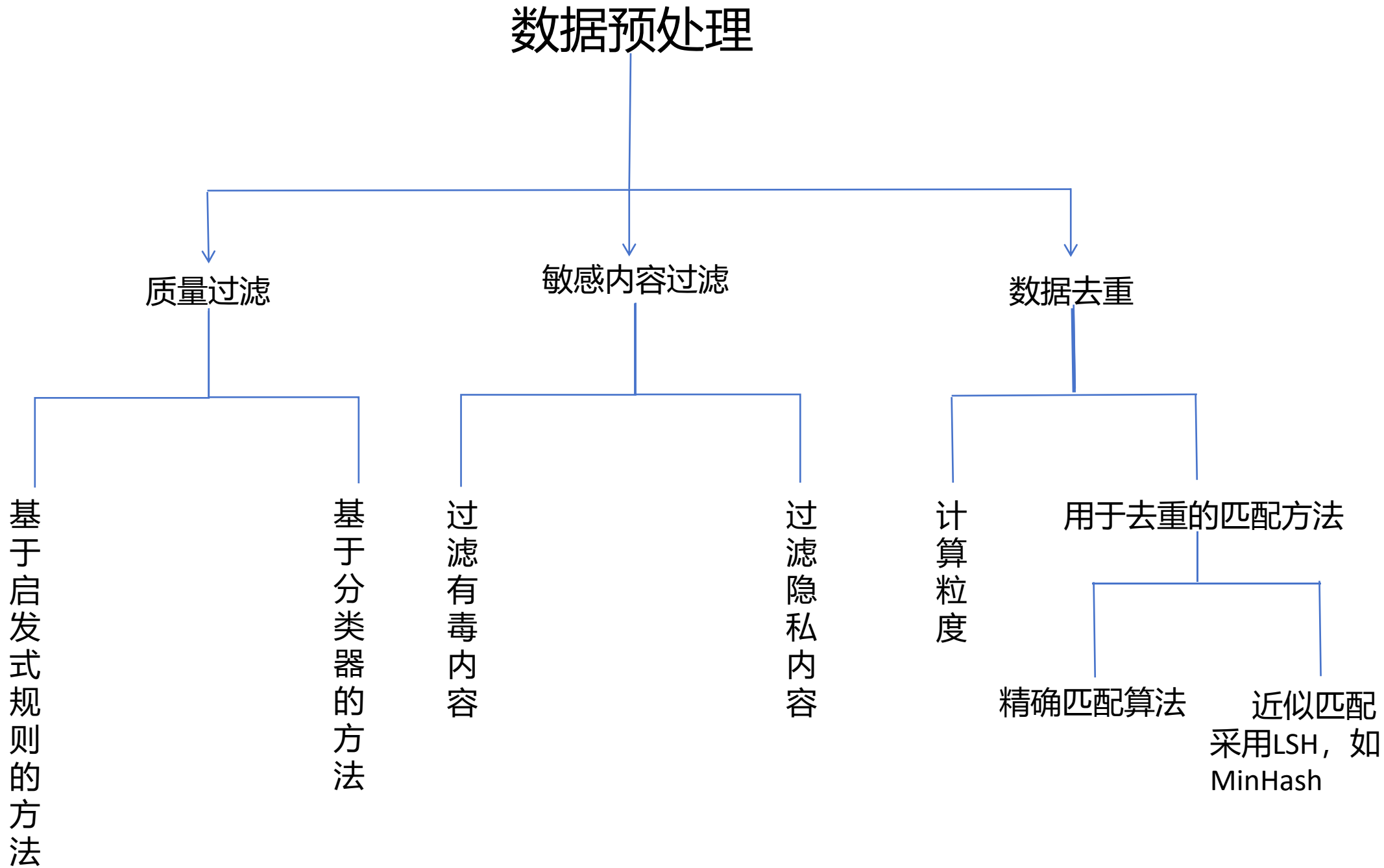
- 过滤有毒内容
- 过滤隐私内容：类似于基于启发式规则，如关键字识别来检测和删除私人信息。

隐私过滤

```python
from utils.rules.regex import REGEX_IDCARD
from utils.cleaner.cleaner_base import CleanerBase

class CleanerSubstitutePassageIDCard(CleanerBase):
    def __init__(self):
        super().__init__()
    def clean_single_text(self, text: str, repl_text: str =
    ↪  "**MASKED**IDCARD**") -> str:
        # 使用正则表达式 REGEX_IDCARD 匹配身份证号，用 repl_text 代替
        return self._sub_re(text=text, re_text=REGEX_IDCARD,
        ↪  repl_text=repl_text)
```

REGEX_IDCARD:正则表达式，目的匹配身份证号码


数据预处理
质量过滤
敏感内容过滤
数据去重
基于启发式规则的方法
基于分类器的方法
过滤有毒内容
过滤隐私内容
计算粒度
用于去重的匹配方法
精确匹配算法
近似匹配
采用LSH，如MinHash

数据去重

重复低质量数据可能诱导模型在生成时频繁输出类似数据，影响模型的性能，也可能导致训练过程的不稳定，训练过程崩溃。

- 计算粒度方法
- 用于去重的匹配方法：精确匹配算法(即每个字符完全相同)；近似匹配算法(基于某种相似性度量),可采用局部敏感哈希(LSH)，如最小哈希(MinHash）。

最小哈希(MinHash)算法

- 它是一种两个集合之间的相似度的技术，其核心思想在于，通过哈希处理集合元素，并选择最小的哈希值作为集合的表示。随后，通过比较两个集合的最小哈希值，来估算出它们的相似度。

- Jaccard相似度：

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

其中：

- $|A \cap B|$ 是集合A和B的交集的大小，即同时属于A和B的元素数量。
- $|A \cup B|$ 是集合A和B的并集的大小，即属于A或B或两者都属的元素数量。

MinHash算法

哈希函数 → 最小哈希签名 → 相似度计算

哈希函数：定义一组不同的哈希函数，每个函数可以随机打乱集合中的元素顺序。

最小哈希签名：对于每个集合，使用多个哈希函数计算其哈希值。记录每个哈希函数下的最小哈希值，形成最小哈希签名。

相似度计算：比较两个集合的最小哈希签名。签名中相同元素的比例即为两个集合的相似度估计。

例子

A={1，2，3}；B={2，3，4}

定义三个简单的哈希函数：

$h_1(x) = (x+1) \mod 5$

$h_2(x) = (2x+3) \mod 5$

$h_3(x) = (3x+1) \mod 5$

计算哈希值：

$h_1(A) = \{2,3,4\}$; 最小值是2

$h_1(B) = \{3,4,0\}$; 最小值是0

$h_2(A) = \{0,2,4\}$; 最小值是0

$h_2(B) = \{2,4,1\}$; 最小值是1

$h_3(A) = \{4,1,4\}$; 最小值是1

$h_3(B) = \{1,4,2\}$; 最小值是1

MinHash签名：

签名(A) = {2, 0, 1}

签名(B) = {0, 1, 1}

计算相似度：

签名相同的位置是第二和第三个位置。

签名相似度 = $2/3 \approx 0.67$

实际Jaccard相似度 = $\frac{2}{4} = 0.5$

句子级去重

```python
import string
import re
from nltk.util import ngrams

class CleanerDedupLineByNgram():
    def __init__(self):
        # 定义行分隔符和元组分隔符
        self.line_delimiter = list("\n")
        chinese_punctuation = "，。！？；“”‘’（）《》【】、|—"
        self.gram_delimiter = list(string.punctuation) + list(chinese_punctuation) + [' ']
    def clean_single_text(self, text: str, n: int = 5, thre_sim: float = 0.95) -> str:
        # 依靠行分隔符分割所有行
        lines = [each for each in re.split('|'.join(map(re.escape, self.line_delimiter)), text) if each != '']
        lineinfo, last = list(), {}
        for idx, line in enumerate(lines): # 计算每行的 n 元组
            # 依靠元组分隔符分割所有 N 元组，并将其暂时存储到 lineinfo 里
            grams = [each for each in re.split('|'.join(map(re.escape, self.gram_delimiter)), line) if each != '']
            computed_ngrams = list(ngrams(grams, min(len(grams), n)))
            lineinfo.append({
                "lineno": idx, "text": line, "n": min(len(grams), n),
                "ngrams": computed_ngrams, "keep": 0
            })

        for idx, each in enumerate(lineinfo): # 过滤掉和相邻行之间 n 元组的 Jaccard 相似度超过 thre_sim 的行
            if last == {}:
                each["keep"], last = 1, each
            else:
                # 计算相邻行间的 Jaccard 相似度
                ngrams_last, ngrams_cur = set(last["ngrams"]), set(each["ngrams"])
                ngrams_intersection, ngrams_union = len(ngrams_last.intersection(ngrams_cur)), len(ngrams_last.union(ngrams_cur))
                jaccard_sim = ngrams_intersection / ngrams_union if ngrams_union != 0 else 0
                if jaccard_sim < thre_sim:
                    each["keep"], last = 1, each
        # 将所有未被过滤掉的 N 元组重新拼接起来
        text = self.line_delimiter[0].join([each["text"] for each in lineinfo if each["keep"] == 1])
        return text
```

thre_sim:相似度阈值

词元化(分词)

- 词元化(Tokenization）旨在将原始文本分割成模型可识别和建模的词元序列，作为大预言模型的输入数据。

- why use?
- 在传统的自然语言处理研究主要使用基于词汇的分词方法，但是基于词汇的分词在某些语

言(如中文分词)中可能对于相同的输入 产生不同的分词结果，导致生成包含海量低频词的词表，还可能存在未登陆词。所以，一些语言模型开始采用字符作为最小单元来分词，目前，子词分词器广泛应用于基于Transformer的语言模型中，如BPE分词，WordPiece分词，Unigram分词。

BPE分词

- BPE算法
- 1.统计频率：统计文本中所有相邻字符对的出现频率
- 2.合并最频繁的字符对：找到最频繁的字符对并将其合并为新的单个字符
- 3.重复上述步骤：反复执行步骤1和步骤2，直到达到预定的词汇表大小或没有更多的字符对可以合并为止。

假设语料中包含了五个英文单词：

"loop"，"pool"，"loot"，"tool"，"loots"

在这种情况下，*BPE* 假设的初始词汇表即为：

["l"，"o"，"p"，"t"，"s"]

在实践中，基础词汇表可以包含所有 ASCII 字符，也可能包含一些 Unicode 字符（比如中文的汉字）。如果正在进行分词的文本中包含了训练语料库中没有的字符，则该字符将被转换为未知词元（如"<UNK>"）。

假设单词在语料库中的频率如下：

("loop"，15)，("pool"，10)，("loot"，10)，("tool"，5)，("loots"，8)

其中，出现频率最高的是"oo"，出现了 *48* 次，因此，学习到的第一条合并规则是（"o"，"o"）→ "oo"，这意味着"oo"将被添加到词汇表中，并且应用这一合并规则到语料库的所有词汇。在这一阶段结束时，词汇和语料库如下所示：

词汇：["l"，"o"，"p"，"t"，"s"，"oo"]
语料库：（"l" "oo" "p"，15），（"p" "oo" "l"，10），（"l" "oo" "t"，10），("t" "oo" "l"，5），("l" "oo" "t" "s"，8）

此时，出现频率最高的配对是（"l"，"oo"），在语料库中出现了 *33* 次，因此学习到的第二条合并规则是（"l"，"oo"）→ "loo"。将其添加到词汇表中并应用到所有现有的单词，可以得到：

词汇：["l"，"o"，"p"，"t"，"s"，"oo"，"loo"]
语料库：("loo" "p"，15)，("p" "oo" "l"，10)，("loo" "t"，10)，("t" "oo" "l"，5)，("loo" "t" "s"，8)

现在，最常出现的词对是（"loo"，"t"），因此可以学习合并规则（"loo"，"t"）→ "loot"，这样就得到了第一个三个字母的词元：

词汇：["l"，"o"，"p"，"t"，"s"，"oo"，"loo"，"loot"]
语料库：("loo" "p"，15)，("p" "oo" "l"，10)，("loot"，10)，("t" "oo" "l"，5)，("loot" "s"，8)

可以重复上述过程，直到达到所设置的终止词汇量。

好处

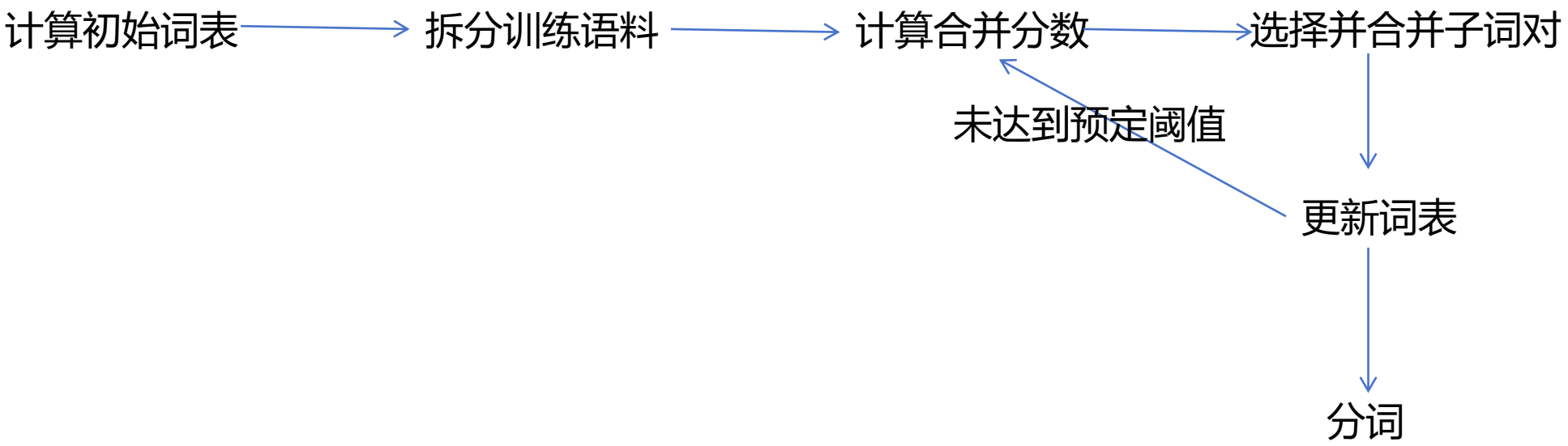
- 将字节视为合并操作的基本符号，实现更细粒度的分割，且解决了未登录词问题。
- 减少词汇表的大小。

WordPiece分词

- WordPiece分词和BPE分词想法类似，都是通过迭代合并连续的词元，但是合并的选择标准略有不同。
- 在合并前，WordPiece分词算法首先训练一个语言模型，并用这个语言模型对所有可能的词元队进行评分，然后，在每次合并时，它都会选择使得训练数据的似然性增加最多的词元对。
  计算公式：

$$\text{score} = \frac{\text{frequency of pair}}{\text{frequency of first element} \times \text{frequency of second element}}$$

WordPiece分词过程



例子

- 假设有如下训练语料中的样例，括号中第2位为在训练语料中出现的频率：

  ("hug", 10)，("pug", 5)，("pun", 12)，("bun", 4)，("hugs", 5)

- 将其拆分为带前缀的形式：

  ("h" "u" "g", 10)，("p" "u" "g", 5)，("p" "u" "n", 12)，("b" "u" "n", 4)，("h" "u" "g" "s", 5)

  这些样例的初始词表将会是：["b", "h", "p", "g", "n", "s", "u"]。

- 计算合并分数：

  对于pair("u", "g")：出现的频率是最高的20次，但"u"出现的频率是36次，"g"出现的频率是20次。所以这个pair("u", "g")的分数是(20)/(36×20)=1/36。同理，计算pair("g", "s")的分数为(5)/(20×5)=1/20。所以，最先合并的pair是("g", "s")→("gs")。

- 此时，词表和拆分后的频率将变成以下：

  Vocabulary: ["b", "h", "p", "g", "n", "s", "u", "gs"]

- 重复上述操作，直到达到想要的词表大小，例如：

  Vocabulary: ["b", "h", "p", "g", "n", "s", "u", "gs", "hu", "hug"]

- 以hugs单词为例：对于单词"hugs"，使用前面示例中的词汇表进行分词，从单词开头开始，在词汇表中能找到的最长子词是"hug"，在这里分割，得到["hug", "s"]。接着处理"s"，发现它也在词汇表中，因此"hugs"的最终分词结果是["hug", "s"]。

Unigram分词

Unigram分词从预料库的一组足够大的字符串或词元初始集合开始，迭代地删除其中的词元，直到达到预期的词表大小。

使用流程

构建初始表 → 计算词元概率 → 选择分词方式 → 优化词表(可选)

构建初始词表：从训练语料中提取所有可能的字符串作为初始词表。子字符串可以是单个字符、字符组合或已有的词汇。

计算词元概率：根据初始词表，计算每个词元在训练语料中出现的概率。

$$\text{词元概率} = \frac{\text{词元的频率}}{\text{词表中所有词元的总频率}}$$

Unigram分词

使用流程

构建初始表 → 计算词元概率 → 选择分词方式 → 优化词表(可选)

分词方式：由于Unigram模型假设词元独立，因此每种分词方式的概率就是词元概率的乘积

优化词表：通过迭代方式优化词表，以减少词表大小并提高分词效果。通常涉及删除一些对整体损失影响较小的词元

例子

假设已有的训练语料和初始表：
训练语料：("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
初始词表：["h", "u", "g", "hu", "ug", "p", "pu", "n", "un", "b", "bu", "s", "hug", "gs", "ugs"]

现对“pug”进行分词：
1.其所有可能的分词方式：["p", "u", "g"] ["p", "ug"] ["pu", "g"]
2.计算每种分词方式的概率：以["p", "u", "g"] 为例
对于["p", "u", "g"]：P(["p", "u", "g"]) = P("p") × P("u") × P("g")
P("p") = 5 / (总频次)
P("u") = 36 / (总频次)
P("g") = 20 / (总频次)
因此，P(["p", "u", "g"]) = (5 × 36 × 20) / (总频次^3)

3.选择概率最高的分词方式：
通过比较，发现["p", "ug"]和 ["pu", "g"]概率相对较高，根据实际需求，进行选取。

在实际应用中，考虑到对于较长文本或复杂的分词任务，可能需使用动态规划算法(如维特比算法)来找到最佳分词路径

维特比算法：

构建模型：先构造一个隐马尔可夫模型(HMM)，该模型包括初始状态概率、状态转移概率和发射概率。

在中文分词任务中，将状态定义为词的开始(B）、词的中间(M）、词的结束(E）和单字词(S），观察值则为句子中的每个字符。

初始化：对于句子中的第一个字符，计算其处于各个状态（B、M、E、S）的概率，并记录到达该状态的最优路径（即前一个状态)。

递归计算：对于句子中的每个后续字符，根据前一个字符的状态和当前字符的观察值，计算当前字符处于各个状态的概率。

终止：在句子的最后一个字符处，找到概率最大的状态（通常是E或S，表示词的结束或单字词)。

回溯：从终止状态开始，根据记录的最优路径回溯，得到整个句子的分词结果。

分词器的选用

- 需要关注的因素
- 分词器须具备无损重构的特性
- 分词器应具有高压缩率

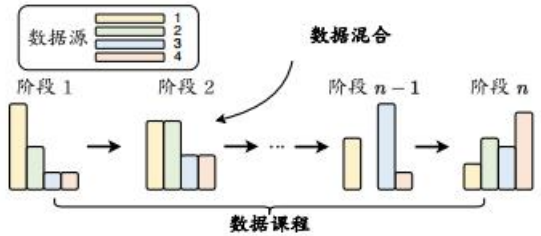
  压缩率公式：

$$\text{压缩率} = \frac{\text{UTF-8 字节数}}{\text{词元数}}.$$

- 其他

数据调度

- 主要关注两方面：
- 各个数据元的混合比例
- 各数据源用于训练的顺序



数据混合

- 在预训练期间，根据混合比例从不同数据源中采样数据：数据源的权重越大，从中选择的数据越多。

常见的几种数据混合策略：

- 增加数据源的多样性
- 优化数据混合
- 优化特定能力
-