分词原理和实践比较

需求分析

利用提供的新浪新闻数据，发现词语。注意，不得利用jieba等第三方自然 语言处理库。可考虑多个n-Gram组合利用。计算出的词汇，需要与jieba等 分词库进行对比研究；

1. 核心目标，探索n-Gram在分词中的价值，结合大数据人工智能技术。该部分需求清楚的点明n-Gram分词在本次作业中的必要性，后提出结合其他人工智能技术，因此本项目还对比了BPE tokenizer。
2. 首先建立特定字集，比如：介词、语气词、时态词、数量词等特定字集，也包括各种标点符号前后的词语，都是一些功能性字，如：说等等；利用上述字，寻找特定组合，如：洗了洗、实际上是洗洗的变体。该部分特定子集即是对中文词汇做描述分析。由于最终分词后得到的分词器包含中文字词字典，因此需要提前了解可能得到的词语的词性，辅助分词器的构建。
3. 形成多个N-Gram比如：2-Gram、3-Gram、4-Gram、5-Gram等，探索利用这些数据，形成词汇。这是一个数据化过程，也是在此基础上的应用； 该部分需求涉及到分词的具体方法，需要将n-Gram和一定长度的滑动窗口结合，涉及到超参数调整。
4. 结果与jieba进行分析对照；即各个分词结果的比较分析，计算量化指标precision，recall，fscore等。

项目说明

本项目以探索分词性能为目标，首先进行语料库的词语分析和数据描述，后从n-Gram分词出发基于滑动窗口实现了词语发现，基于迭代算法、字典树、有向无环图和动态规划实现了Ngram分词器。随后介绍和比较了字符级别BPE技术和字节级别BPE技术，最终拓展到大语言模型分词器的思考。

重点难点分析

1. 本项目的重点是基于一定的算法给出自己的分词方法，方法包括训练词表，得到词表、对测试语句进行分词。
2. 本项目的难点是分词器的效果需要足够的好，即能够：
    a. 合理利用训练文本高效构建合适的词表
    b. 对测试文本进行恰当地分词，正确处理未见词汇（oov）问题
    c. 比较多种分词器的特点和效果得出一定的可泛化结论

操作手册

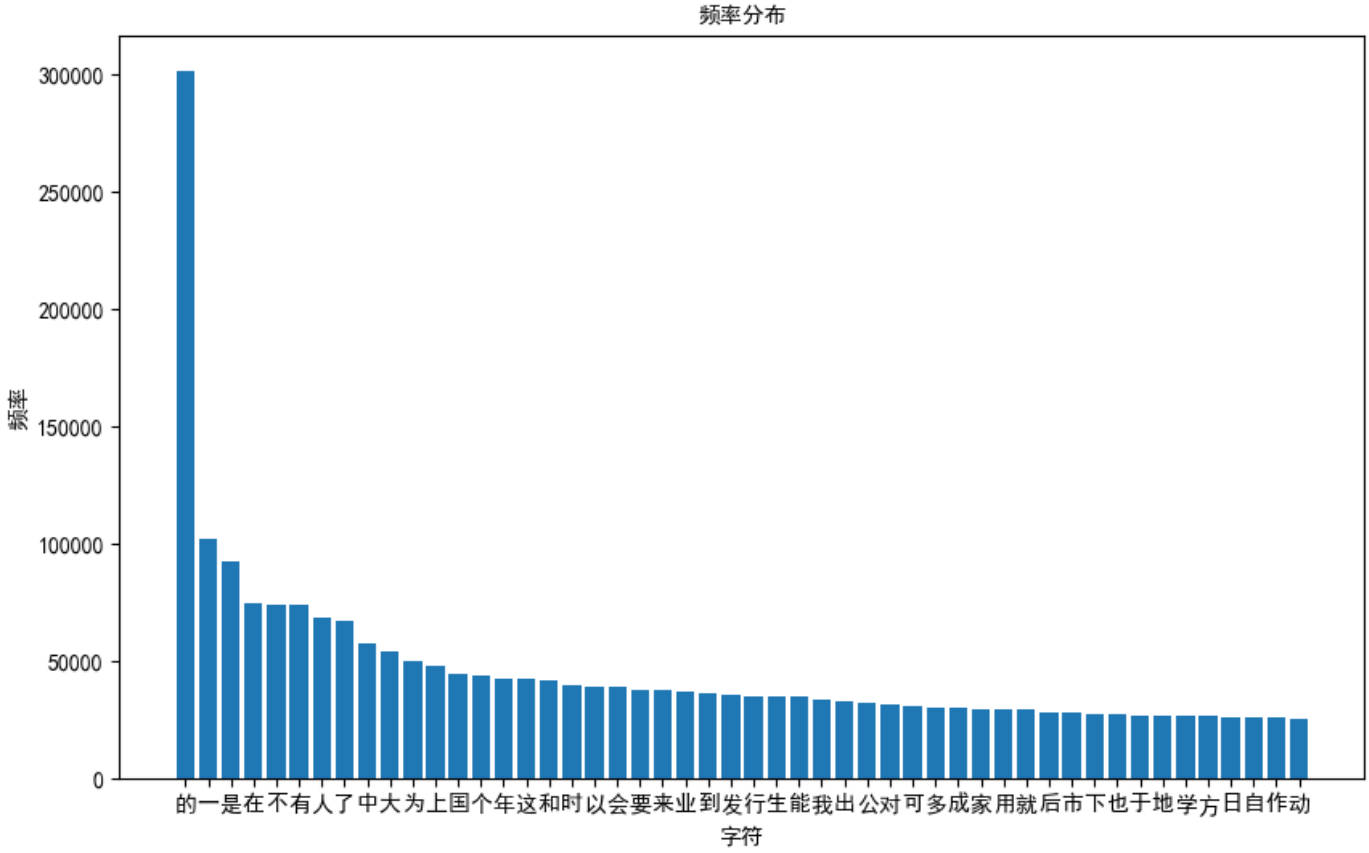
该部分介绍代码文件夹的结构。

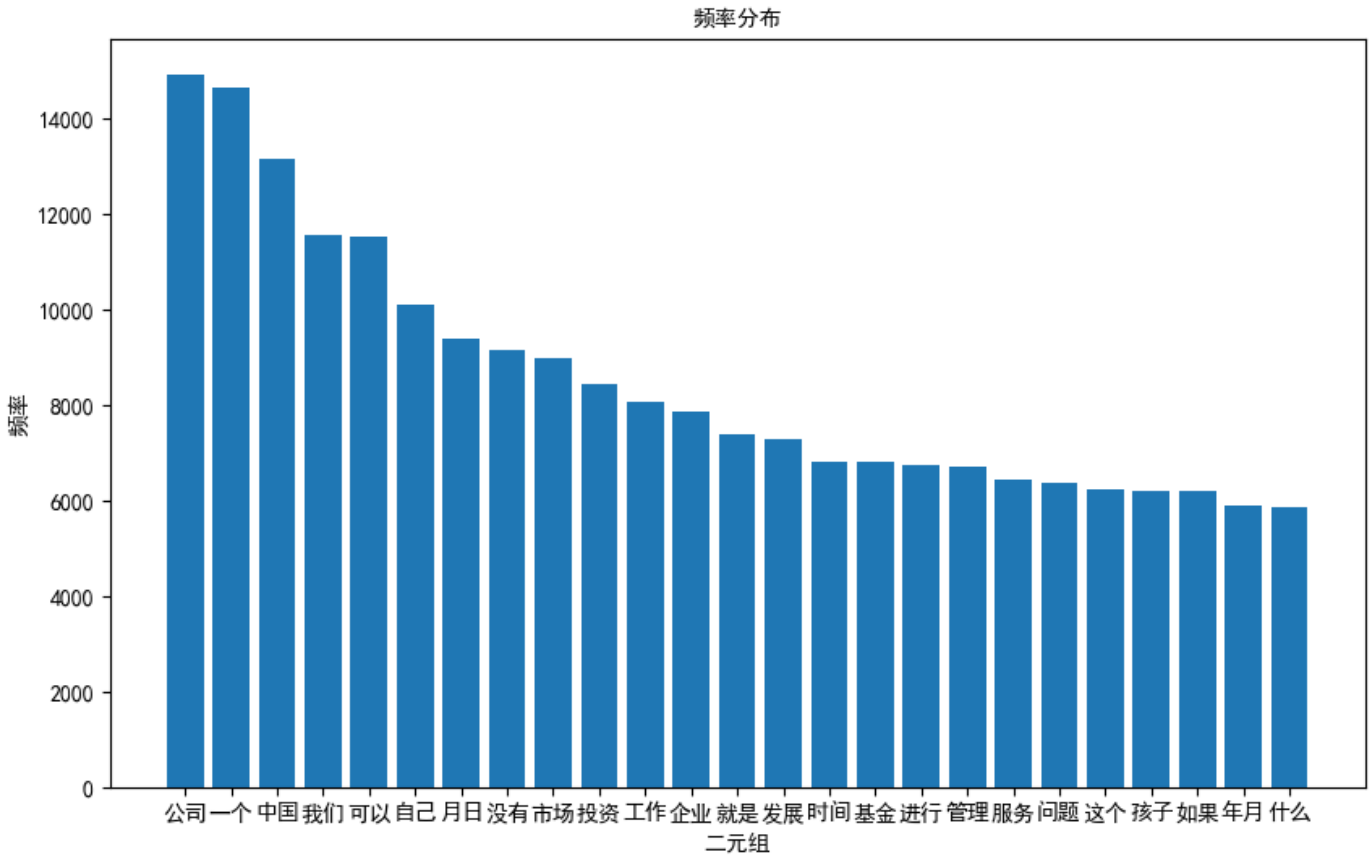
```
computer_final/
├── build_vocab.py
├── evaluate.py
├── forward_cut.py
├── load_data.py
├── max_prob_cut.py
├── train.py
├── visualization.py
├── results/
├── visualization/
├── bpe/
│   ├── bbpe_tokenizer.py
│   ├── bpe_chn_tokenizer.py
│   └── bpe_eng_tokenizer.py
├── model/
│   ├── trans_dict.model
│   ├── trans_dict_2.model
│   ├── word_dict.model
│   └── word_dict_2.model
└── README.md
```

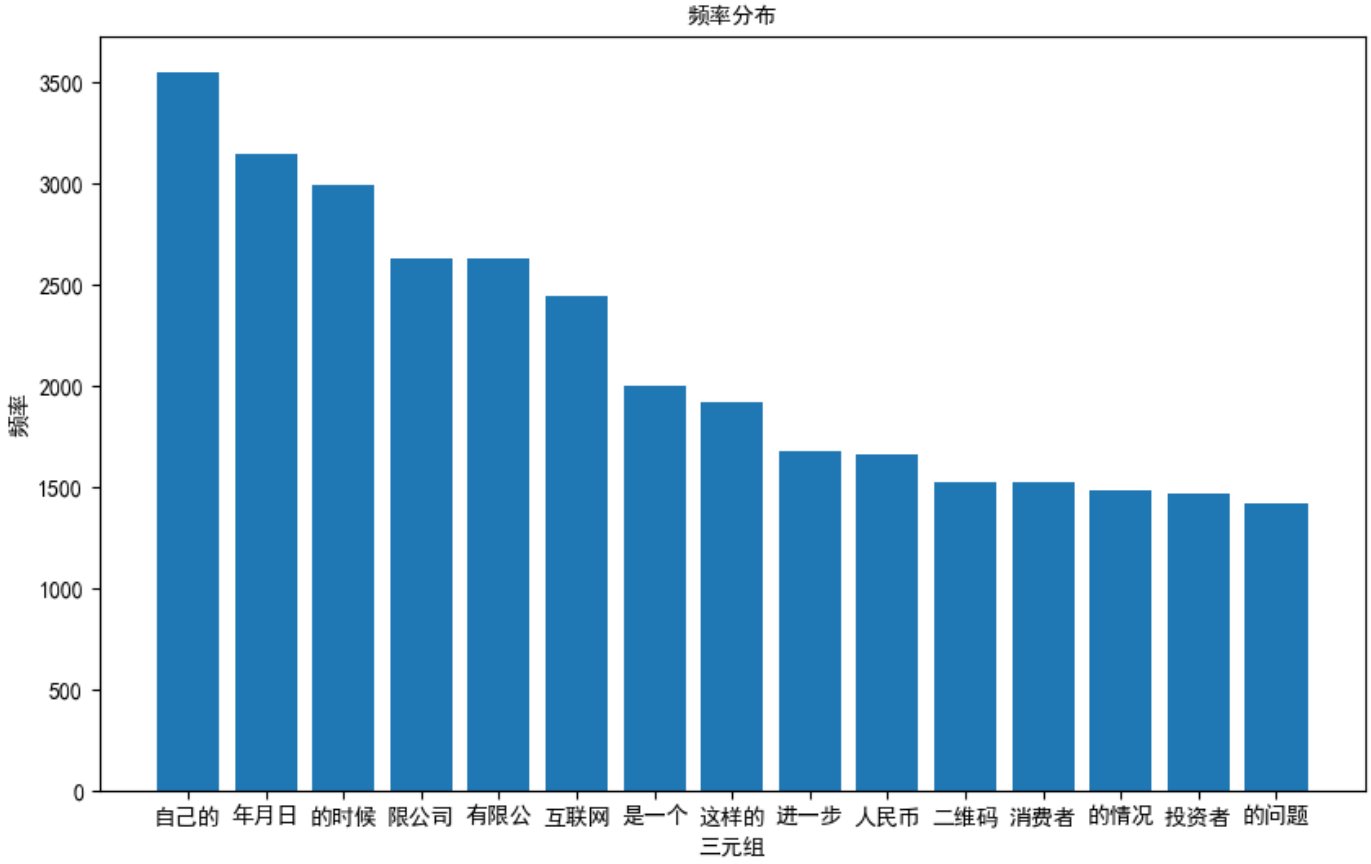
N-gram分词方法

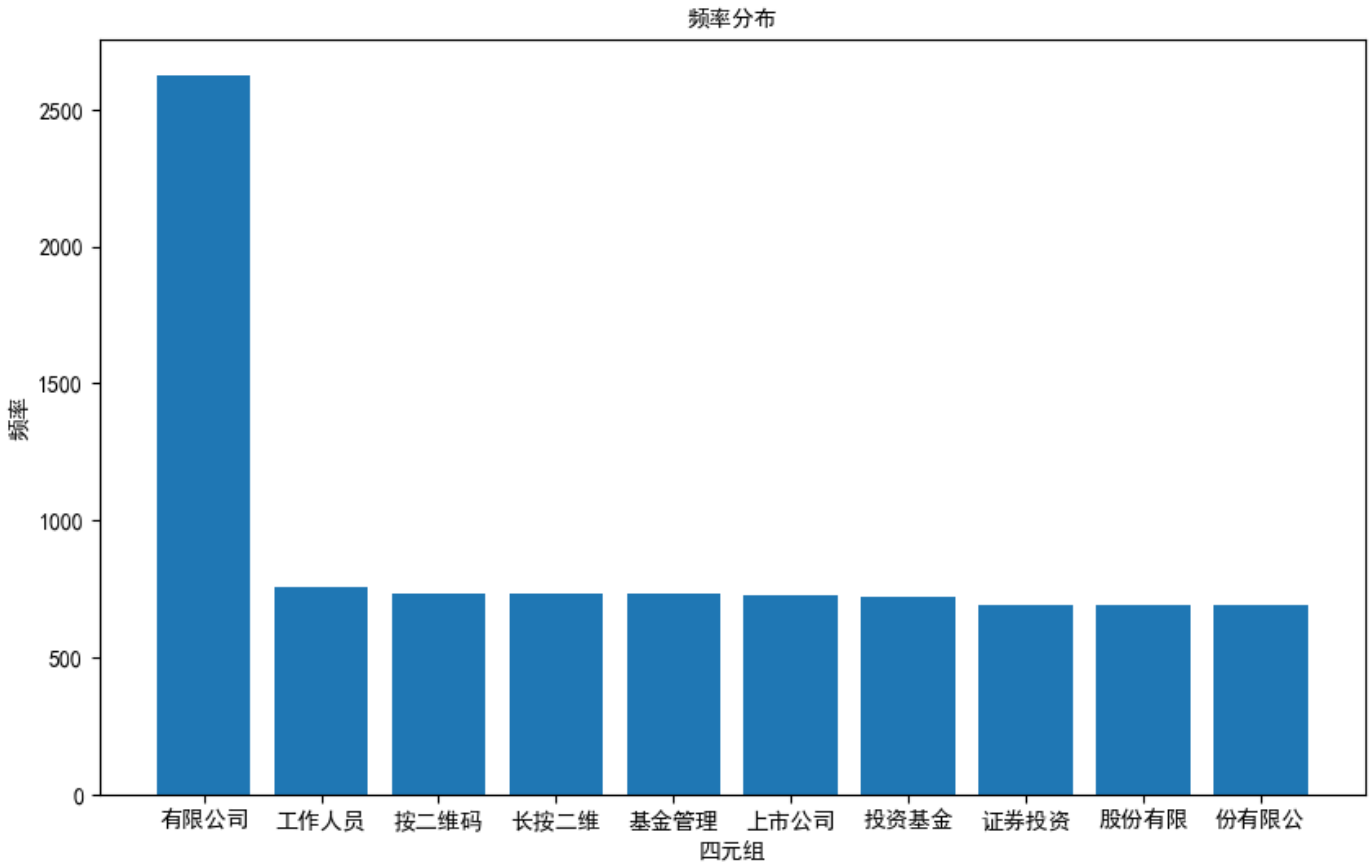
数据描述

1. 首先结合汉语的词语习惯，统计一到四元组的出现频率，展示高频组合。在统计前去除了特殊符号、中英文标点、数字等非汉字符号。具体结果如下：


频率分布
300000
250000
200000
150000
100000
50000
0
频率
的一是在不有人了中大为上国个年这和时以会要来业到发行生能我出公对可多成家用就后市下也于地学方日自作动
字符


频率分布
14000
12000
10000
8000
6000
4000
2000
0
频率
公司 一个 中国 我们 可以 自己 月日 没有 市场 投资 工作 企业 就是 发展 时间 基金 进行 管理 服务 问题 这个 孩子 如果 年月 什么
二元组





可以发现存在大部分没有实际意义的字符组合，例如“的”，“这个”，“就是”，“自己的”等情况。于是建立特定的子集对该类组合进行模式识别。

2. 为发现中文中词汇的构成模式，先建立介词、语气词、时态词、数量词等特定字集。后基于特定字集以及标点符号，寻找其前后常见的组合。得到结果部分展示如下：

	前面最多的词	后面最多的词	中间最多的词	两端最多的词
为	因（为）	（为）了	（为）最（为）	5（为）5
了	到（了）	（了）一	（了）解（了）	拍（了）拍
对	针（对）	（对）于	（对）方（对）	面（对）面
！	了（！）	（！）免	（！）这（！）	/

基于滑动窗口的词语发现

首先基于滑动窗口进行词语发现得到词表，后基于该词表对句子进行简单的最大前向匹配得到基本的分词结果。

1. 词语发现
    a. 为了构建词典，借鉴字符级别的BPE编码的思想。即先初始化词典为全体单个字符，后采用当前词典对句子进行最大前向匹配得到词语串。随后统计全体句子中相邻连续两个词语二元组（随着迭代进行其每个元素长度不定，可能是1也可能是2等）的出现次数。统计完毕后，针对那些频率较高的词语二元组做合并，对词表进行更新。随着迭代的进行词表中最大长度的词语会越来越长，形成一些高频的多字词。
    b. 在本步骤中，随着迭代进行，合并的高频二元组比例应当逐渐下降才合理。在第一次合并得到二元词语时，将二元组的前80%进行合并，在第二次何合并得到二元词语或三元词语时，将高频二元组的前40%进行合并；第三轮合并时将高频二元组的前27%进行合并。最终最大词语长度为6.最终词表中二元词语占比11.06%，三元词语占比1.04%，四元词语占比0.27%。得到的部分词语如下：

二元词语	鹦鹉	蹒跚	前排	憧憬
三元词语	趵突泉	氧化钇	红蜻蜓	吃空饷
四元词语	心存侥幸	同仇敌忾	汗流浃背	铤而走险

2. 词典对比
    a. 为了对比词典效果，比对https://github.com/liuhuanyong/WordSegment.git中给出的词典。值得注意的是，上述步骤中发现的词语有将近50%并未在该开源词典中出现（展示的例子都是没有在开源词典中出现的），即本方法很好地实现了词典的补充。
    b. 但是由于训练文本大多为微博新闻语料，词汇有所局限，为了提升词典的质量，此处合并了开源词典以提升后续分词的效果。

N-gram tokenizer

1. 算法介绍

a. n-gram模型，即N元模型，适用于中文分词。该模型假设第n个词的出现仅与前n-1个词相关，与其他词无关，整个句子的概率即为各词出现概率的乘积。这些概率可通过统计语料中相关词同时出现的频率来计算。常见的模型有Bi-gram和Tri-gram模型。

b. 基本原理

   i. 假设一句话由m个词组成，则该句子的概率可以表示为m个词语按顺序出现的联合概率 $P(w_1, w_2, \cdots, w_m)$.根据概率论中的链式法则得到如下：
   $P(w_1, w_2, \cdots, w_m) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdots P(w_m \mid w_1, w_2, \cdots, w_{m-1})$

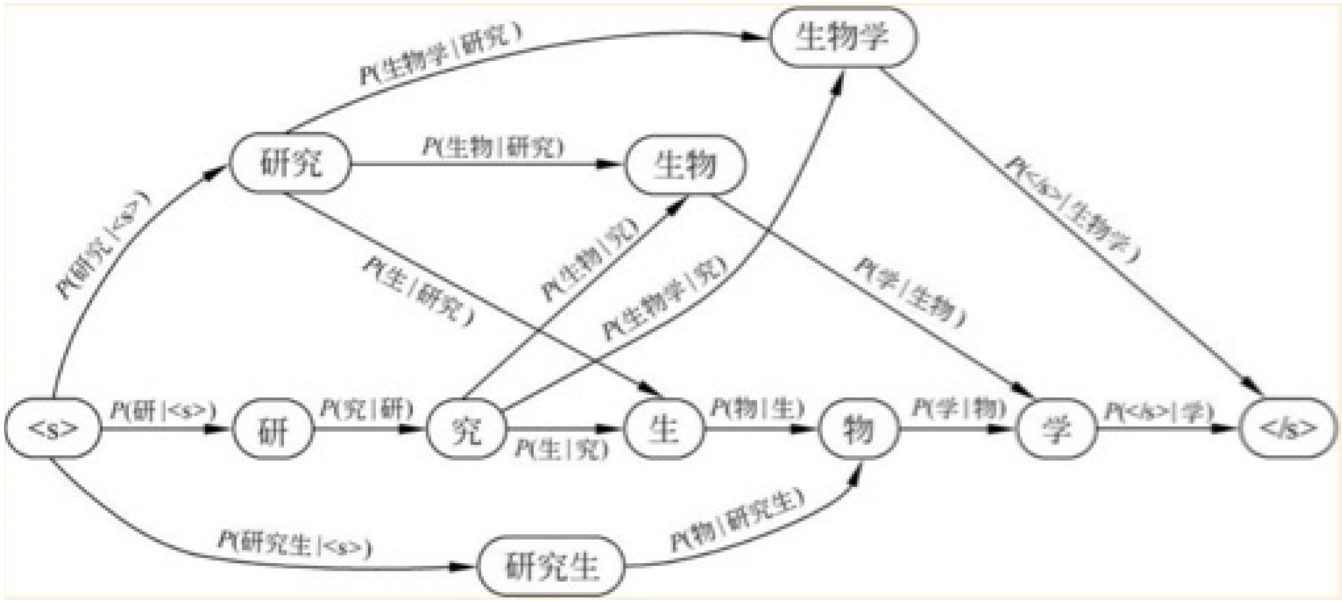
   ii. 直接计算这个概率的难度有点大，根据n-gram的假设，当前词仅与前面n-1个词相关，即 $P(w_i \mid w_1, w_2, \cdots, w_{i-1}) = P(w_i \mid w_{i-n+1}, \cdots, w_{i-1})$ 。其中i为某个词的位置。当n取1时，可以得到 $P(w_1, w_2, \cdots, w_m) = \prod_{i=1}^{m} P(w_i)$ ；当n取2时，可以得到 $P(w_1, w_2, \cdots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-1})$ ；当n取3时，可以得到 $P(w_1, w_2, \cdots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-2}, w_{i-1})$.

   iii. 其中该句子中由n-gram假设得到的 $P(w_i \mid w_1, w_2, \cdots, w_{i-1}) = P(w_i \mid w_{i-n+1}, \cdots, w_{i-1})$ 概率可以由频率近似概率，即找到语料库中全体出现($w_{i-n+1}, \cdots, w_{i-1}$)的个数n1，并统计n1个情况中后一词为n2的个数n2，并相除得到 $P(w_i \mid w_{i-n+1}, \cdots, w_{i-1})$ 的近似估计。若n1或n2存在0，则采用平滑方法近似处理。

   iv. 对于由n个汉字组成的一句话，根据不同的切法可能可以得到不同的分词效果。n-gram分词方法的思想是最大似然，即选取的切法得到的m个词语所计算得到的句子出现概率 $P(w_1, w_2, \cdots, w_m)$ 是最大的，其中句子概率的计算方法会结合n-gram假设进行简易计算。

c. 算法组件:为了高效地存储和计算，n-gram分词常采用字典树存储句子，采用有向无环图存储各种分词方式，使用动态规划方法求解最大的分词方式。

   i. 有向无环图：对于某个要切分的句子如“研究生物学”，如果句子有n个汉字，n-1个间隔，则一共有 $\sum_{i=0}^{n-1} C_{n-1}^{i}$ 个切分方式。全体切分方式可以表达为一个有向无环图，其中节点是全体切分方式下得到的所有词汇，只有在句子中相邻出现的词语节点之间才会建有向边，以Bigram方法为例，node1->edge->node2的有向边上的值就是转移概率P(node2|node1)。最优的分词方式就是从<s>节点走到</s>节点的累计转移概率最大的路径。而计算每条边上的概率值要结合字典树进行估计。在Bigram下，每个节点是一个词，但是在n-gram下，每个节点内是n-1个词语组成的序列。



ii. 字典树：字典树是一种嵌套字典结构，可以用于快速的前缀匹配和查找词语。前文提到由于bigram方法需要计算条件概率P(node2|node1)，则需要快速查找给定前缀出现的总次数 $n(node_1) = \sum_{node_i} n(node_1, node_i)$ 以及给定前缀下对应的下一字符频数 $n(node_1, node_2)$ 。字典树则是一种十分适合的数据结构。需要注意的是，我们并不需要采用字典树把一整个句子的词语从头到尾嵌套表示在字典树中，只需要把所有出现过的n-gram存储在字典树即可。即字典树的深度（按照某条路径下键的个数计）最大为n。

iii. 动态规划：为了求解DAG的最大路径，需要使用动态规划方法高效求解。以Bigram方法下的上图为例，如果用R(node)代表从起点<s>出发到达node节点的最大累计概率，则目的是找到一条路径使得R(</s>)最大。

   1. 转移方程：想要走到</s>下累计概率最大则需要从其全体可能的前驱节点（图中是两个前驱节点<生物学>和<学>）当中选择一个作为出发点。选择的前驱节点prenode即是 $argmax[R(prenode) + weight(prenode, </s>)]$
   2. 重复子问题：为了得出上方的最优prenode，要进一步求解全体prenode的R(prenode)，则得到了子问题。

   最终通过从后往前的递归计算我们可以得到最优路径，即n-gram方法下的最优分词。

iv. 已有词表以及训练集处理：

   1. 已有词表：由于ngram方法在训练阶段需要针对已经分词好的句子统计词频和n元组的共现次数，因此还需要根据已有的词表对训练集句子先做切分。举个例子，在测试阶段针对上方“研究生物学”的例子做分词，所得到的有向无环图中不会有<s>到<研究生物>或是<s>到<研究生物学>这一条路径，是因为ngram方法在训练阶段使用的词表并没有<研究生物>和<研究生物学>这两个token。通过构建词表可以极大程度减轻测试阶段可能的切分情况，以及训练阶段统计二元组字典树的复杂程度。在本项目中，词表的构建主要通过上方滑动窗口发现词语所进行。

2. 训练集处理：如同机器学习方法一样，ngram方法在训练阶段需要已知训练数据的最佳切分方式，并根据句子切分成的词语列表进行n元组统计，最后应用于测试集上没有分词的连续句子。在本项目中拿到了的训练文本并没有做分词处理，为了避免提前应用jieba等成熟的分词器导致引入数据泄露偏差，因此选择在得到了词表后采用简单的最大前向匹配对训练集进行处理。即对于训练数据结合自行发现的词表做最大前向匹配分词得到词语列表，后进行字典树构建。

d. 算法流程：构建词表——对训练文本分词——构建字典树——对测试文本构建有向无环图——基于动态规划对测试文本分词

2. 效果展示：

a. 测试集：首先对测试数据文本进行分词，下面展示一些分词结果。

```
1 新风 系统 的 主要 用途 是 室内 外 的 通 风 换 气 但 不仅仅 是 通 风 换 气 它 可
  以 排出 室内 污染 有害 空气 同时 将 室外 富 氧 空气 过 滤 后 送 入 室内 无数 事
  实 证明 许多 疾病 癌症 白 血 病 呼吸 系统 疾病 等 都 与 室内 空气 污染 有关 新风
  系统 可 有效 解决 室内 空气 污染 的 问题 改善 室内 的 空气 环境 确保 家人 的 健
  康 呼吸 米 亚 新风 系统 换 气 不仅仅 是 排 去 污染 的 空气 除了 有 换 气 功能 外
  还 具有 除 臭 除尘 排 湿 调节 室 温 的 功能 米 亚 新风 系统 换 气 功能 排出 被 污
  染 的 空气 供 给 人们 呼吸 所 需要 的 新鲜 空气 让 室内 小时 都 保持 舒适 畅通
  米 亚 新风 系统 除 臭 功能 换 气 扇 能 迅速 排出 各种 原因 引起 的 不适 的 臭 味
  制造 一个 舒适 的 环境 米 亚 新风 系统 除尘 功能 漂 浮 在 空气 中 的 灰 尘 附 有
  许多 肉 眼看 不到 的 细 菌 所以 要 驱 走 居 室 工作 场所 里 的 尘 埃 创造 一个 舒
  适 的 环境 米 亚 新风 系统 排 湿 功能 居 室 里 的 湿 气 不仅仅 来自 浴 室 人体
  和 燃 具 也 会 释放 出 水分 而且 建筑 密 闭 性 更 好 易 出现 暖 房 等 因 结 露
  而 发 霉 床 和 墙 被 腐 烂 的 问题 所以 用 换 气 扇 经常 除去 室内 的 湿 气 能
  使 居 室 和 人 保持 舒适 和 健康 米 亚 新风 系统 调节 室 温 夏天 的 夜晚 用 换
  气 扇 驱 走 室内 的 热气 把 外面 凉 爽 的 空气 替 换 进来 冬天 进行 全 热 交换 减
  少 室内 温度 流失 提高 冬天 的 取 暖 效果 米 亚 新风 系统 附加 功能 内 循环 功能
  关闭 室外 进 风 通道 进行 室内 循环 实现 急 速 室内 净 化 旁 通 功能 春秋 季 室
  内 温 差 较 小时 室内 外 空气 不 经过 热 回 收 芯 体 实现 直 进 直 出 空气 流通
  净 化
2
```

b. 回测训练集：随后将训练好的分词器用于训练数据集，用于在训练集上比对最开始的最大前向分词和ngram分词的结果。经过统计，有20.6%句子分词前后不相同，取出一些例子做展示和分析：

```
1 ngram_round1:食 尚 玩 家 中央 公园 爱 的 溜冰
2 max_forward: 食 尚 玩 家中 央 公园 爱 的 溜冰
3
4 ngram_round1:消灭 星星 完 美 中文 版 开 心 消 消 乐 连连 看 连 一 连 过 家家
5 max_forward: 消灭 星星 完 美中 文 版 开 心 消 消 乐 连连 看 连 一 连 过 家家
```

```
6
7 ngram_round1:记者 从 今天 召开 的 国防部 例 行 记者 会上 获悉 中国 将 于 今年 月 在 北京 举行 阅 兵 式 按照 国际 惯例 中国 军队 邀请 参加 过 中国 人民 抗日战 争 世界 反 法西斯 战争 东方 战场 作战 的 国家 以及 其它 有关 国家 军队 派员 参加 阅 兵 式 并 出席 观 摩 活动
8 max_forward: 记者 从今 天 召开 的 国防部 例 行 记者 会上 获悉 中国 将 于 今年 月 在 北京 举行 阅 兵 式 按照 国际 惯例 中国 军队 邀请 参加 过 中国 人民 抗日战 争 世界 反 法西斯 战争 东方 战场 作战 的 国家 以及 其它 有关 国家 军队 派员 参加 阅 兵 式 并 出席 观 摩 活动
```

可以发现，尽管原始训练数据按照最大前向算法切分，但是训练得到的n-gram分词器在相同训练数据上实现了自我提升，有20.6%的数据在第一轮ngram的结果下表现更好。这启发我们可以进行迭代训练，基于第一轮ngram分词器继续对原始训练数据做分词，重新构建字典树和转移字典训练第二轮ngram分词器。

3. 结果评估：先提前展示表格——

序号	1	2	3	4	5
数据集	训练集	训练集	测试集	测试集	测试集
方法	最大前向切分	ngram_round1	最大前向切分	ngram_round1	ngram_round2
precision	0.5807	0.581	0.5946	0.594484	0.594565
recall	0.7455	0.746	0.7552	0.756103	0.756259
fscore	0.6529	0.6533	0.6654	0.665674	0.665684

   a. 同样可以评估原始的训练数据集采用最大前向分词相比于jieba分词的效果。其中，原始的最大前向分词精确率0.5807，召回率0.7455，f值0.6529，结果记录在第1列。采用第一轮ngram分词方法再重新在训练数据上进行分词，结果记录在第2列，精确率0.581，召回率0.746，f值0.6533.通过表格第1列和第2列的对比，可以发现通过第一轮训练得到的ngram分词器在训练数据上效果好于最大前向分词。即前文提到过的“自我提升”现象的出现，启发我们可以进行迭代训练。

   b. 为了对比本ngram分词和jieba分词的效果差异，将jieba分词视为正确分词结果，计算测试集上的precision,recall以及f1-score。结果如下：精确率0.5945，召回率0.7561，f值0.6657.结果记录在第4列。考虑到训练数据并没有切分，词表发现方法属于自监督，且jieba分词是一个非常成熟的分词器（依托动态规划寻找分词轨迹），它效果的差距可以接受。后评估测试数据集上采用前向最大分词方法在jieba分词作为标准分词结果下的效果，结果记录在第3列.通过3/4列的对比可以发现ngram分词法有所提升。

4. 迭代进行

a. 在得到了第一轮ngram分词器后，可以进一步接着对原始训练数据进行最大概率分词。由于新的ngram分词器的分词效果会比先前的最大前向分词效果更好，因此用更好地分词器处理的更高质量的训练数据能够得到一个更佳的字典树和转义字典，从而进一步得到效果更好的ngram分词器。整个流程迭代进行，会逐步提升ngram分词器的效果。碍于计算资源，本项目仅进行2轮迭代。

b. 第二轮训练后的ngram分词器在测试集上进行测试，并将jieba分词的结果作为ground truth计算得到precisison为0.5945，recall为0.7563，fscore为0.6657记录于第5列。具有轻微的提升。

字符级别BPE分词方法

BPE算法介绍

字节对编码（BPE, Byte Pair Encoder）是一种数据压缩算法，常常用于在构建英文分词器。其可以实现在固定大小的词表中凑成可变长度的子词，还可以有效处理OOV问题。该算法简单有效，因而目前它是最流行的方法（GPT, GPT-2, RoBERTa, BART, LLaMA, ChatGLM）。

字符BPE的英文分词

在英文分词场景下，BPE 首先将词（如apple)分成单个字符（‘a','p','p','l','e','<\w>'，其中'<\w>'代表单词末尾)，初始tokens字典就是这些字母和'<\w>'。然后依次用另一个字符替换频率最高的一对字符（如经过全体语料库统计'a''p'出现次数最多则合并'a''p'），直到循环次数结束。字符级别的BPE算法常适用于英文等欧美语言拉丁语系，因为此类语言一般只有有限个字符（如英文26个字母，因为单字节编码即可表示英语），可以用有限的字节进行对应。此时字符和字节可以近似等价，所以字节对编码退化到字符级别的字节对编码。

1. 训练步骤：

   a. 第一步：在所有单词后面加<\w>token并统计所有单词的出现次数辅助下一步统计字符频数

   b. 第二步：设定最大subword个数V，在V次循环中，每次都统计每个连续字符对的出现频率，从中选择**最大**的那个出现次数**非1**的字符对进行合并。例如某次循环中要合并’e‘和'st'因为'est'这一连续字符对出现的频率在所有连续字符对中最大为count，则新增{’est‘:count}进入词表而'e'和'st'对应频数都减count（若减为0则从词表中去除）。反复进行。需要注意随着迭代的进行，每一轮只会合并最大的那一个连续字符对，因此往往不会得到过于生僻的新subword；且每一轮迭代会新增一个新subword（但同时可能减少0或1或2个subword，因为原始字符的频次会降低可能从词典中弹出），则V轮迭代后最多新增V个新词，可以有效控制词典大小。

```
==========
Tokens Before BPE
Tokens: defaultdict(<class 'int'>, {'l': 7, 'o': 7, 'w': 16, '</w>': 16,
'e': 17, 'r': 2, 'n': 6, 's': 9, 't': 9, 'i': 3, 'd': 3})
```

```
Number of tokens: 11
==========
Iter: 0
Best pair: ('e', 's')
Tokens: defaultdict(<class 'int'>, {'l': 7, 'o': 7, 'w': 16, '</w>': 16, 'e': 8, 'r': 2, 'n': 6, 'es': 9, 't': 9, 'i': 3, 'd': 3})
Number of tokens: 11
==========
Iter: 1
Best pair: ('es', 't')
Tokens: defaultdict(<class 'int'>, {'l': 7, 'o': 7, 'w': 16, '</w>': 16, 'e': 8, 'r': 2, 'n': 6, 'est': 9, 'i': 3, 'd': 3})
Number of tokens: 10
==========
Iter: 2
Best pair: ('est', '</w>')
Tokens: defaultdict(<class 'int'>, {'l': 7, 'o': 7, 'w': 16, '</w>': 7, 'e': 8, 'r': 2, 'n': 6, 'est</w>': 9, 'i': 3, 'd': 3})
Number of tokens: 10
==========
Iter: 3
Best pair: ('l', 'o')
Tokens: defaultdict(<class 'int'>, {'lo': 7, 'w': 16, '</w>': 7, 'e': 8, 'r': 2, 'n': 6, 'est</w>': 9, 'i': 3, 'd': 3})
Number of tokens: 9
==========
Iter: 4
Best pair: ('lo', 'w')
Tokens: defaultdict(<class 'int'>, {'low': 7, '</w>': 7, 'e': 8, 'r': 2, 'n': 6, 'w': 9, 'est</w>': 9, 'i': 3, 'd': 3})
Number of tokens: 9
==========
```

2. 对新句子编码：BPE的编码过程较为简单，即贪婪策略。在之前的算法中，我们已经得到了subword 的词表，对该词表按照字符个数由多到少排序。编码时，对于一个句子用空格划分得到的每个单词，遍历排好序的子词词表寻找是否有 token 是当前单词的子字符串，如果有，则该 token 是表示单词的 tokens 之一。我们从最长的 token 迭代到最短的 token，尝试将每个单词中的子字符串替换为 token。 最终，我们将迭代所有 tokens，并将所有子字符串替换为 tokens。 如果仍然有某个单词的子字符串没被替换但所有 token 都已迭代完毕，则将剩余的子词替换为特殊 token，如<unk>.

```
# 给定单词序列
["the</w>", "highest</w>", "mountain</w>"]

```

```
# 排好序的subword表
# 长度 6         5           4        4         4       4          2
["errrr</w>", "tain</w>", "moun", "est</w>", "high", "the</w>", "a</w>"]

# 迭代结果
"the</w>" -> ["the</w>"]
"highest</w>" -> ["high", "est</w>"]
"mountain</w>" -> ["moun", "tain</w>"]
```

3. 解码：将所有的 tokens 拼在一起即可

```
# 编码序列
["the</w>", "high", "est</w>", "moun", "tain</w>"]

# 解码序列
"the</w> highest</w> mountain</w>"
```

字符BPE的中文分词

中文场景下，情况变得复杂。因为常用的汉字有5000多个，且中文的每个字无法像英文单词那样拆成subword或是letter，因此字符级别的BPE算法不适合中文分词，需要采用字节级别的BPE编码。但是有一些方法巧妙地做了转换。此处做简单的展示。其思想就是把一个中文句子看成是“英文句子”，把每个中文汉字看成是“英文字母（字符）”，在中文句子的末尾添加'<\w>'，则先前合并字母得到子词的过程可以复用到合并汉字得到词语的过程。正是借鉴这样的思想，本项目在ngram分词中进行了词语发现。此处不再做例子展示，相关代码在文件夹内。

字节级别BBPE分词方法

BBPE算法介绍

1. 前面提到，字符级别的BPE算法只能适用于字符像英文这样字符个数有限的语言，难以适用于中文这样字符个数多的语言。此外一般的大语言模型分词器处理的输入可能涉及到多个语言，因此需要一个能够兼容多种语言的分词器。2019年12月，论文：Neural Machine Translation with Byte-Level Subwords 在基于BPE基础上提出以Byte-level为粒度的分词算法Byte-level BPE，即BBPE。其具有其独特的优势，可以很好地解决OOV问题，而且是一种无损压缩方法，即完全还原输入文本。
2. 在介绍BBPE之前，先简单介绍下相关的基础概念。

a. **Unicode** 是一种字符集，它为每个字符分配了一个唯一的数字编号（码点），例如“中”的码点是 U+4E2D。Unicode 的目的是**为世界上所有字符**提供统一的编码标准，以解决不同国家和地区字符编码不兼容的问题。在表示一个Unicode的字符时，通常会用“U+”然后紧接着一组十六进制的数字来表示这一个字符。如下图。如果想知道某个字符的unicode，在python中使用内置函数ord()即可，例如ord('A') 得到65（注意Unicode是10进制表示的）；反向操作函数chr()。

U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4e00	一	丁	丂	七	丄	丅	丆	万	丈	三	上	下	丌	不	与	丏
4e10	丐	丑	丒	专	且	丕	世	丗	丘	丙	业	丛	东	丝	丞	丟
4e20	丠	両	丢	丣	两	严	并	丧	丨	丩	个	丫	丬	中	丮	丯
4e30	丰	丱	串	丳	临	丵	丶	丷	丸	丹	为	主	丼	丽	举	丿
4e40	乀	乁	乂	乃	乄	久	乆	乇	么	义	乊	之	乌	乍	乎	乏
4e50	乐	乑	乒	乓	乔	乕	乖	乗	乘	乙	乚	乛	乜	九	乞	也
4e60	习	乡	乢	乣	乤	乥	书	乧	乨	乩	乪	乫	乬	乭	乮	乯
4e70	买	乱	乲	乳	乴	乵	乶	乷	乸	乹	乺	乻	乼	乽	乾	乿

b. **UTF-8** 是一种针对 Unicode 的可变长度字符编码，它根据码点的大小，将其编码为 1 到 4 个字节（Unicode十进制数转字节序列）。这里的8是指最小单位是1字节。

   i. UTF-8 的编码规则如下：

      - U+0000 - U+007F: 1 个字节
      - U+0080 - U+07FF: 2 个字节
      - U+0800 - U+FFFF: 3 个字节
      - U+10000 - U+10FFFF: 4 个字节

      例如，“中”的码点是 U+4E2D，它属于第三行，被编码成三个字节：11100100 10111000 10101101。再比如『汉』这个字的Unicode编码是0x6C49。0x6C49在0x0800-0xFFFF之间，使用3字节模板：1110xxxx 10xxxxxx 10xxxxxx。将0x6C49写成二进制是：0110 1100 0100 1001， 用这个比特流依次代替模板中的x，得到：11100110 10110001 10001001。

   ii. UTF-8相比于其他的字符编码方式具有可变长度的优点。UTF-32对于每个字符都采用4位字节（byte）过于冗长。而UTF-8编码是一个变长的编码，有1～4个范围的字节(bytes)长度。对于不同语言中字符采用不同长度的字节编码，例如英文字符基本都是1个字节（byte）。中文汉字通常需要2～3个字节，大多是3个byte，一个 byte 用 int8 表示。

c. **Unicode 和 UTF-8 的关系：**简单来说，**Unicode 是字符集**，而**UTF-8 是字符集的一种编码规则**（除了UTF-8还有UTF-16等）。Unicode 规定了每个字符的码点，而 UTF-8 规定了如何将这些码点转换为字节序列进行存储和传输。UTF-8保证任何语言都可以通用。Unicode转换为UTF-8的字节序列如下图所示。

Unicode编码(十六进制)	UTF-8 字节流(二进制)
000000-00007F	0xxxxxxx
000080-0007FF	110xxxxx 10xxxxxx
000800-00FFFF	1110xxxx 10xxxxxx 10xxxxxx
010000-10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

字符在不同编码方式下的字节序列如下图。

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

3. 合并过程：通过上面的介绍我们知道，想要处理多语言分词，可以将每个语言的字符转为Unicode进行处理。由于UTF-8编码的变长编码方式更加高效，因此选择将字符转成UTF-8编码后对字节序列进行处理。随后再采用BPE算法的思想进行subword合并。
    a. 在GPT2论文中提到如果使用全体UTF-8的字符编码（变长1～4位字节编码）作为基础单元去构建词表时，未融合前词表就有超过130000个不同Token过多了。因此Byte-level BPE只用单个UTF-8字符编码中的单个字节，即原始词表只有256个不同的编码作为词表中基础的单元去做BPE的融合过程，经过融合后最终GPT2的词表仅仅只有50000多个Token。按照先前类似的思想对这个词表进行合并，例如先前字符级别的BPE可能某一轮合并的是's'和't'，现在BBPE字节级别某一轮合并的可能是字节序列‘00110000’和‘01000010’。下面是论文中展示的合并结果：

Original		質問して__証明と証拠を求めましょう	Ask__questions,__demand__proof,__demand__evidence.
Byte		E8 B3 AA E5 95 8F E3 81 97 E3 81 A6 E2 96 81 E8 A8 BC E6 98 8E E3 81 A8 E8 A8 BC E6 8B A0 E3 82 92 E6 B1 82 E3 82 81 E3 81 BE E3 81 97 E3 82 87 E3 81 86	41 73 6B E2 96 81 71 75 65 73 74 69 6F 6E 73 2C E2 96 81 64 65 6D 61 6E 64 E2 96 81 70 72 6F 6F 66 2C E2 96 81 64 65 6D 61 6E 64 E2 96 81 65 76 69 64 65 6E 63 65 2E
BBPE	1K	E8 B3 AA E595 8F しE381 A6 __E8 A8 BC 明 E381 A8 E8 A8 BC E6 8B A0 をE6 B1 82 めE381 BE しょう	A s k __quest ions , __dem and __pro of , __dem and __ev idence .
	2K	E8 B3 AA 問 しE381 A6 __E8 A8BC 明 E381 A8 E8 A8BC E68B A0 を E6 B1 82 めE381 BE しょう	A s k __qu est ion s , __d em and __pro o f , __d em and __e v id ence .
	4K	E8 B3 AA 問 しE381 A6 __E8 A8BC 明E381 A8 E8 A8BC 拠 をE6 B1 82 めE381 BE しょう	As k __quest ions , __d em and __pro of , __d em and __ev id ence .
	8K	E8 B3 AA問 しE381 A6 __E8 A8BC 明E381 A8 E8 A8BC 拠 をE6 B1 82めE381 BE しょう	As k __questions , __demand __pro of , __demand __evidence .
	16K	E8 B3 AA問 しE381 A6 __E8 A8BC 明E381 A8 E8 A8BC 拠 をE6 B1 82めE381 BE しょう	As k __questions , __demand __proof , __demand __evidence .
	32K	E8 B3 AA問しE381 A6 __E8 A8BC 明E381 A8 E8 A8BC 拠 をE6 B1 82 めE381 BE しょう	As k __questions , __demand __proof , __demand __evidence .
CHAR		質 問 し て __ 証 明 と 証 拠 を 求 め ま し ょ う	A s k __ q u e s t i o n s , __ d e m a n d __ p r o o f , __ d e m a n d __ e v i d e n c e .
BPE	16K	質問 して __ 証明 と 証拠 を求 め ましょう	As k __questions , __demand __pro of , __demand __evidence .
	32K	質問 して __証明 と 証拠 を求め ましょう	As k __questions , __demand __proof , __demand __evidence .

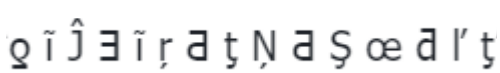
Figure 5: An example from Ja-En tokenized with different vocabularies. Raw spaces are replaced by underscores and spaces are used to split tokens. We can observe how tokens look like as the tokenization granularity goes from fine to coarse: Byte (256) → BBPE (1K, 2K, 4K, 8K) → Char (8K) → BBPE (16K, 32K) → BPE (16K, 32K).

关注到红色箭头，输入是日文当词表大小是1K时，A8 BC 两个字节并没有合并成一个 Token，在词表大小是2K时，A8BC被BBPE合并成一个Token。同理我们观察到在 1K时，E595 8F 没有被合并，在2K时被合并成 日本 "聞" 。也就是随着词表大小的增大，会有越来越多的新词被合并。（但是需要注意的是，在合并的时候来自语言的token的字节序列并不会合并出不存在的跨语言新token）

b. 同时关注上图中Byte层，当输入Original是不同语言（日本，英语）时，在字节（byte) 层面有一定几率会出现相同的一段字节(byte)编码，从某总程度上具有一定迁移性。同时也正是由于字节层面比字符粒度更低一层，也会导致在解码的过程中对于某个字节不确定是来自某个Character还是单独的Character中从而导致歧义。这个时候可能需要借助上下文的信息和一些动态规划的算法来进行解码。

4. 编码过程：

a. 在编码时，先将给定的中文句子每个汉字用字节序列表示。一般每个汉字的unicode在UTF-8编码下为3个字节。“我爱你中国”每个汉字用三个字节表示得到15个字符见下图。为方便渲染，下方15个字符简写为‘abcdbfghigklgnh’.（注意第二和第五个字符重复出现，第七和第十个和第十三个字符重复出现，第八和第十五个字符重复出现）然后再根据合并规则选取需要合并的连续字符进行合并，合并完把连续字符转换为最终的token id。



b. 在合并时，和字符级别的BPE算法一样，按照合并规则的优先级（词频递减顺序）选取合并规则中的subword，然后贪心地对unicode字符串（如“a b c d b f g h i g k l g n h”）做全局匹配，即把所有可以合并的相邻字符做合并。最终合并完可能是“abc dbf ghi gkl gnh”，则可以轻易解码为“我 爱 你 中 国”。

c. 最后把合并完的字符串映射为id，则得到了token id.

d. 问题：由于合并是字节级别，缺乏字符级别信息，可能存在一些其他合并情况，如“abc d bf ghi gkl gnh”这样的编码（合并）结果。那么要是直接解码，“d”或是“bf”都无法转换为“爱”，因为两者没有合并，朴素解码并不知道“d”其实属于后者“爱”这一汉字，有可能会把“d”或是“bf”解码成对应的其他字符，甚至如果极端情况下还会存在一些合并的字节序列对应不合法的字符。于是需要下方的动态规划解码策略。

5. 解码时的动态规划

a. 解码过程是把编码后的序列还原为中文的过程。显而易见，并不是所有的字符序列都可以还原为中文，识别的结果可能会有一些无效的组合，所以需要采用基于动态规划的最佳解码方法。论文提出的最优子结构和转移方程为如下。即对于某个字节序列 $\{B\}_{k=i}^{j}$，解码的目的是将其转换为尽可能多的有效字符，并设前k个字节序列可以得到的字符个数为 $f(k)$，$g(k-t+1，k)$ 代表着 $\{B\}_{k-t+1}^{k}$ 序列是否是一个有效字符，如果是则为1否则为0.

$$f(k)=\max_{t=1,2,3,4}\{f(k-t)+g(k-t+1,k)\}$$

在递归计算f(k)的过程中，同样会记录每个k位置下选择的最优切分方法，则最终回溯即可得到最优的解码策略。如果某子序列即使在合并后仍旧无法对应任何字符，则不会被解码成字符，即跳过了非法字符。

b. 举个几个例子：

i. 如果”我爱你中国“在编码阶段合并为“abc dbf ghi gkl gnh”，则k取5（从1起），f(5)=1+f(4)，因为最后的字节序列“gnh”可以对应“国”；以此类推解码为“我 爱 你 中 国”；

ii. 如果“我爱你中国”在编码阶段合并为“abc d bf ghi gkl gnh”，那么k=6(k从1起),f(6)=1+f(5)=1+1+f(4)=1+1+1+f(3)；
f(3)=max{f(2)+g(3,3),f(1)+g(2,3),f(0)+g(1,3)}=f(1)+g(2,3)，因为g(2,3)对应“爱”（假设“abcd”不对应合法字符，“d”不对应合法字符，“abcdbf”不对应合法字符）。最后f(1)=1对应“我”。最终也能得到“我 爱 你 中 国”。

iii. 如果某个大模型自回归生成的tokenid经过转换变为了“abc bf ghi gkl gnh”，即在第二个位置从“dbf”变成了“bf”，那么在动态规划的解码策略下，后三个字节序列得到“你 中 国”，但是f(2)=max{f(1)+g(2,2)=1+0,f(0)+g(1,2)=0+0}，因此第二个字节序列被忽略。最终解码得到“我 你 中 国”。

iv. 因此可见，解码部分的动态规划会根据编码阶段已经合并后的字节序列做进一步的合并（即编码解码都合并字节），从而尽可能生成完整字符，避免某些未合并完整的字节序列解码为非法字符。论文指出，由于UTF-8编码的特殊性，该动态规划转移方程求解的编码策略是唯一的。

c. 论文提出的动态规划策略与ngram下的最大累计概率动态规划策略既有相同点也有不同点。相同点在于都基于动态规划，都是从后往前回溯进行，都依赖于一个初始词表，转移方程有所类似。区别如下：

   i. ngram动态规划是在分词阶段，也就是编码阶段把句子分成离散的token id时进行的。但是BBPE是在解码阶段已经有了token id后做合并时进行的。

   ii. ngram动态规划的目的是寻找转移概率最大，因此需要存储前缀树构建有向无环图等，转移方程需要带入token间的转移概率（浮点类型）。但BBPE的动态规划目的是解码的字符最多，基于字节序列，转移方程需要带入某字节序列是否是合法字符（布尔类型）。

   iii. ngram动态规划需要得到多种分词策略后再保留概率最大的那个，计算量大；但BBPE分词只有唯一的解码策略。

拓展思考

- Python中使用什么函数查看字符的Unicode，什么函数将Unicode转换成字符？用ord()查看一个字符的Unicode码点，是一个十进制数。用chr()把一个十进制数转换为unicode下对应的字符。
- Tokenizer的vocab size大和小分别有什么好处和坏处？
  - size大好处：更高的压缩率，更多的子字节序列合并为新的token，则encode一句话所需要更少的token，提高大模型吞吐率；更强的语义表达能力，例如只有把“苹”“果”合并为“苹果”，大模型才可能理解它是一个整体概念甚至可能是手机。而不单单是一种叫“苹”的水“果”；更长的文本处理能力：在固定token生成个数或是上下文token个数限制之下，如果词表更大则平均而言每个token的代表的语义信息更多，则模型可以输入和生成的token序列可以解码为更长更多的自然语言。一个极端的例子就是语料库中每个句子都是一个token。
  - size大坏处：内存占用；限制泛化能力和稀有字符匹配，例如处理拼写错误比较困难，用户拼错的小众单词很难在词表中出现，因为每次合并都是合并高频subword，且还可能导致较短的subword弹出vocab导致难以用以配对别的小众单词。
  - size小好处：更广泛的单词生成能力（例如26个英文字母就生成了很多单词）；节省内存；适合多语言任务（例如用户自己造的法语英文混杂单词）；更好处理生僻词或拼写错误语法错误。
  - size小的坏处：增加计算成本（极端情况下一个token是一个字节序列，那么一个句子会被编码的很长很长，中文平均会翻三倍的长度）；减少上下文范围（如前文所说）；
- 为什么LLM不能拼写单词？为什么LLM认为9.11比9.9大？为什么不能数strawberry里有几个r? 因为这些字符被tokenized为token，一个token是一些字节或字节序列，可能对应了多个字母或是一个字母都不包括。LLM的建模以token为基础，但是不知道每个token对应什么字母或对应几个字母。
- 为什么 LLM 不能处理非常简单的字符串操作任务，比如反转字符串？
  - LLM采用BBPE分词器，每个字符被拆成了多个字节，因此LLM反转字符需要保证字符倒序下每个字符内部的字节还要维持原序，加大了难度。

    - 即使成功做到了字符倒序且字符内部的字节原序，在解码时的动态规划策略只能单向从后往前解码。
    - LLM是最大似然训练出来的概率模型，推理和逻辑能力欠缺。如果预训练数据集缺乏相关练习题，则很难表现的很好。
- 为什么 LLM 在非英语语言（例如日语）上表现较差？tokenizer在非英语数据上训练不足。
    - 一方面是训练数据大多为英文的原因
    - 另一方面是因为非英语语料转为token后长度扩大很大（如中文每个汉字三个字节），导致训练起来常超过上下文限制等。例如：
        - 句子一：“Originated as the Imperial University of Peking in 1898, Peking University was China’s first national comprehensive university and the supreme education authority at the time. Since the founding of the People’s Republic of China in 1949, it has developed into a comprehensive university with fundamental education and research in both humanities and science. The reform and opening-up of China in 1978 has ushered in a new era for the University unseen in history. And its merger with Beijing Medical University in 2000 has geared itself up for all-round and vibrant growth in such fields as science, engineering, medicine, agriculture, humanities and social sciences. Supported by the “211 Project” and the “985 Project”, the University has made remarkable achievements, such as optimizing disciplines, cultivating talents, recruiting high-caliber teachers, as well as teaching and scientific research, which paves the way for a world-class university.”被BBPE编码成id序列后长度只有原来的0.72倍
        - 句子二：“博士学位论文应当表明作者具有独立从事科学研究工作的能力，并在科学或专门技术上做出创造性的成果。博士学位论文或摘要，应当在答辩前三个月印送有关单位，并经同行评议。学位授予单位应当聘请两位与论文有关学科的专家评阅论文，其中一位应当是外单位的专家。评阅人应当对论文写详细的学术评语，供论文答辩委员会参考。”被BBPE编码后长度却是原来字符串长度的2.97倍。
- 为什么 LLM 在简单算术问题上表现不好（9.11和9.9谁大）？因为数字的划分很随机。如677被GPT-2划分为2个token，而不是3个数字。
- 为什么 GPT-2 在编写 Python 代码时遇到比预期更多的困难？在一定程度上是建模的问题，涉及模型架构、数据集和模型参数，但也涉及到tokenization，因为可能对空格的编码效率太低。例如在代码中每个token都是一个token，针对生成和训练时都需要频繁地输入输出同一个id。而在自然语言中，Egg位于句子开头可以划分为一个token，但是” egg“即空格加egg则变为一个token。这种冲突也增加了麻烦。（而对于GPT-4，它的分词情况就好很多，对Python中空白字符的处理有了很大改进；所以从CPT-2到GPT-4的Python编码能力的提高不仅仅是语言模型、架构和优化细节的问题，而且也来源于分词器的设计以及它如何将字符组合成token）

- 为什么 LLM 遇到字符串 “<|endoftext|>” 时会突然中断？没有对应处理特殊token的逻辑。例如生成答案中本来就需要<|endoftext|>而不是答案末尾时，output_ids中会有多个<|endoftext|>对应的id。但是tokenizer的解码方法遇到第一个<|endoftext|>就会立刻停止解码。
- 为什么当问 LLM 关于 “SolidGoldMagikarp” 的问题时 LLM 会崩溃？“SolidGoldMagikarp”实际上是一个Reddit用户，训练tokenizer的数据集与实际训练llm的数据集非常不同。在分词数据集中，可能有大量的Reddit数据，而“SolidGoldMagikarp”是其中一个经常发帖的人，这个词出现的频率很高，所以在分词器中被合并成一个单独的token处于词表中。但是在LLM的预训练数据中，经过预训练数据清洗会把无意义的用户名去除，即“SolidGoldMagikarp”等字符串没有在Reddit相关的预训练数据中出现。这个token对应的嵌入向量在优化的开始是随机初始化的，并且在模型训练的过程中从未被更新过（缺乏相关的预训练语料）。在测试的时候，如果你输入的文本含有”SolidGoldMagikarp“会直接被分词为一个token，则context中加入了一个embedding matrix中从没有被更新过的一行，接着再自回归生成就会导致输出token不可控效果低。【启发在预训练造语料时要确保tokenizer里的所有token要出现】
- 为什么在使用 LLM 时应该更倾向于使用 YAML 而不是 JSON？JSON被BBPEtokenizer转token时生成的tokens非常密集，而YAML在更高效一些。例如YAML可以直接表示一个整数123，而不需要像JSON那样将其表示为字符串“123”，json当中“”以及{}等都需要拆成字节进行编码。
- 为什么 LLM 实际上不是端到端的语言建模？严格意义上讲，LLM的BBPEtokenzier会把字符做进一步的拆分，这个预处理的步骤使得LLM不是直接对于自然语言做处理，这也导致了很多“9.11比9.9大”的LLM乌龙。LLM其实是一个token id到token id的语言建模，而不是一个自然语言到自然语言的建模。

参考连接

ngram

https://www.cnblogs.com/xlturing/p/8467021.html

https://blog.csdn.net/Zh823275484/article/details/87878512

https://codle.net/chinese-word-cutter-1/

https://www.cnblogs.com/CocoML/p/12725988.html

https://lujiaying.github.io/posts/2018/01/Chinese-word-segmentation/

BPE

https://wmathor.com/index.php/archives/1517/

https://blog.csdn.net/a1097304791/article/details/122068153

https://blog.csdn.net/watermelon_c/article/details/139333172

https://blog.csdn.net/thomas20/article/details/137194169

https://zhuanlan.zhihu.com/p/424631681

https://www.less-bug.com/posts/using-bpe-principle-for-chinese-word-segmentation-plate/

BBPE

https://zhuanlan.zhihu.com/p/137875615

https://zhuanlan.zhihu.com/p/146114164

https://pku-llm.ai/course/24fall/homework1/

https://zhuanlan.zhihu.com/p/649030161

https://blog.csdn.net/weixin_36378508/article/details/134883614

https://zhuanlan.zhihu.com/p/664717335

https://arxiv.org/pdf/1909.03341

https://blog.csdn.net/watermelon_c/article/details/139333172