- 拼音输入法
  - 原嘉锐 2022010756
  - 实验环境
  - 使用的语料库和处理方法
    - 语料库选择
    - 词频统计
  - 二元模型拼音输入法
  - 三元即以上模型拼音输入法
  - 其他合理的评价指标
  - 其他思考和尝试
    - 采用不同数据库进行训练
    - 句首字的概率统计
    - 使用大模型的API进行尝试
    - 几种方法的正确率比较
  - 建议和感受

拼音输入法

原嘉锐 2022010756

实验环境

windows 11 系统 , python版本为3.10,需要安装的包见requirements.txt,可以在pwoershell或者cmd中运行，建议使用conda 虚拟环境

使用的语料库和处理方法

语料库选择

使用sina_news_gbk和webtext2019zh(后简称为sinanews和webtext),分别为新闻语料和社区回答语料，原因在于考虑前者有较多专业名词后者则会包含更多生活上的用语，可能会使得模型具有更好的泛化能力,对于第二个数据库以及二者混合的效果见 其他思考与尝试部分,对于二元和三元算法的讨论部分仅呈现使用第一个数据库的效果

词频统计

- 二元： 对于二元的情况，考虑存成一个“总字数”*“总字数”的numpy,并在统计以后存储为pkl的格式减小文件大小,这样存虽然存在表中会有很多无用的0的情况,但是numpy可以支持更快

的访问和修改速度，从而提升词频统计和后续搜索的速度
- 三元: 对于三元的情况，使用numpy在运行时会占至少40G的内存,显然无法运行，所以改为存储{"三元词":频数}的键值对,并最终存储为pkl的文件格式

二元模型拼音输入法

- 思路与公式 对于输入的一串拼音，从首字开始计算使得$\Pi_{i=1}^{n}P(w_i|w_{i-1})$最大的句子，其中$P(w_i|w_{i-1})=\frac{N(w_{i-1}w_i)}{N(w_{i-1})}$,考虑到有些情况下$w_{i-1}w_i$并不会出现，故做平滑化处理，采用$R(w_i|w_{i-1})=\lambda_1 P(w_i|w_{i-1})+\lambda_2 P(w_{i-1})$来代替上述中的$P(w_i|w_{i-1})$,对于求解过程,采用基于动态规划的viterbi算法,即求解
- $$Q(W_{ij})=\begin{cases} max_k(Q(W_{i-1,k})*R(w_{ij}|w_{i-1,k})) & i\neq 0 \\ P(w_{i,j}) & i=0 \end{cases}$$
- 并在每一步求解时存储使得当前Q最小的k值,从而可以在最后往前迭代输出整个句子
- 实验效果

1. 在结果中当$\lambda_1=1$,$\lambda_2=0$时可以达到0.397的句准确率和0.854的字准确率
2. 训练耗时大约2.5min，即统计词频的时间
3. 生成全部501条输入总共耗时2.8左右,平均每句耗时0.006s左右

- 案例分析

1. 数据库分析由于数据库语料是新闻语料这一特点,可以看到诸如 青山绿水就是金山银山,夺取新时代中国特色社会主义伟大胜利等新闻中常见词都可以回答正确,但是向 阳光开朗大男孩则会变成 阳光开朗大南海,这种极具网络口语化的内容则可能正确率不高,且人名 特朗普正确,但是 郑和错误，这也和语料有很大关系.
2. 三元词分析二元模型对于一些三元的词汇会存在错误的情况,比如 清华慈真正成为中国瓷器的主流中的 青花瓷一词则出现错误,此问题可以在三元模型中获得解决.
3. 首字分析 二元中出现了 尼加我的微信,这种错误在我看来与首字的概率赋值有很大关系,事实上对于首字我觉得应该重新统计该字作为句首出现的概率,该部分实验在 其他思考与尝试部分
4. 多音字分析 由于语料库是没有注音标记的语料库,所以最终得到的模型很难区分多音字,模型会把 第一输出为 的一,因为 的的出现频率很高,同时由于 某某人的一个某某物品这种结构的句子也很常见,所以会出现这种情况

- 调参 调参能带来的提升对于二元来说比较有限,故没有花太多时间在调参上,以下为手动尝试的几组参数和结果,在三元的部分使用了optuna协助调参

$\lambda_1:\lambda_2$	句准确率	字准确率
1:1	0.836	0.325

$\lambda_1:\lambda_2$	句准确率	字准确率
9:1	0.858	0.383
100:1	0.857	0.391
1000:1	0.856	0.395
10000:1	0.856	0.397
100000:1	0.856	0.397
1:0	0.854	0.397

可以看到随着二元概率占比的增加巨准确率逐步上升,且最后基本稳定在0.397

- 复杂度分析

1. 空间复杂度,由于存储的是一个2维的numpy,所以空间复杂度为n^2，其中n为词典的大小,约为6000左右
2. 时间复杂度,使用二元的viterbi算法，时间复杂度为nm^2，其中n为输入的句子长度,m为输入的拼音中对应字最多的字数

三元即以上模型拼音输入法

- 思路与公式 三元与二元模型其实区别不大,实际上四元以及更高其使用viterbi算法的逻辑都几乎没有什么差别,实现上也几乎一致,只需要更改数据处理和公式即可,三元的R和viterbi需要搜索的Q定义如下

$$R(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i)$$

$$Q(W_{ij}) = \begin{cases} max_{k,m}(Q(W_{i-1,k}) * R(W_{ij}|W_{i-2,m}W_{i-1,k})) & i \in 0,1 \\ P(W_{ij}) & i = 0 \\ \lambda_2 max_k(Q(W_{i-1,k})(P(W_{ij}|W_{i-1,k}) + \lambda_3 P(W_{ij}))) & i = 1 \end{cases}$$

但实际使用后发现,这样的计算方法似乎提升较为不明显,经过初步的调参后发现句准确率大概在 0.45左右,经过一些分析和输出发现对于 人工智能的 工智能这个三元词语,最优输出位 攻智能，因为 攻智一次一共出现6次,而 攻智能也是6次，相比之下 工智出现5400多次,而 工智能则出现5300多次,算出来的概率反而前者更大,故考虑稍微修改公式

$$R(w_i|w_{i-2}w_{i-1}) = \lambda_1 N(w_i w_{i-2}w_{i-1}) * P(w_i|w_{i-2}w_{i-1}) + \lambda_2 N(w_i w_{i-1}) * P(w_i|w_{i-1}) + \lambda_3 P(w_i)$$

其余并无大变化,取得了较大的提升

- 案例分析

1. 语料库分析这种由于语料库限制导致的错误依旧存在 量实施农民伯伯辛勤劳动换来的,不难发现 量实施这种词极具新闻语料特色
2. 三元的优势 相比于二元，一个典型的例子是 青花瓷真正成为中国瓷器的主流中的三元词 青花瓷输出正确

- 调参

$\lambda_1:\lambda_2:\lambda_3$	字准确率	句准确率
1000:8:1	0.899	0.527
50000:20:1	0.908	0.558
80000:20:1	0.908	0.560
500000:900:26(optuna调参得到的最优)	0.910	0.567

- 复杂度分析

1. 存储使用key:value的键值对,空间复杂度较大,可以使用稀疏矩阵的形式来进行存储,可以使得存储空间压缩至$\frac{1}{5}$左右
2. 时间复杂度,搜索算法的时间复杂度为nm^3因为相比于二元在搜索时需要遍历之前的两层,跑完所有500个句子需要耗时200s左右，耗费时间较长

其他合理的评价指标

- 首字正确率 由于viterbi搜索时，第一个字是无法使用动态规划得到的,所以如何规定首字的情况具有较为重要的意义，相应地首字正确率也具有较为重要的意义,经过测试二元的首字正确率为0.834,字准确率为0.854，三元模型的首字正确率为0.88，字准确率为0.91，可以看到首字正确率低于字正确率,处理好首字具有较为重要的意义

其他思考和尝试

采用不同数据库进行训练

- 二元的情况(均采用调参得到的最优比例1:0 仅sinanews

```
word right: 0.854757338840537
sentence right: 0.3972055888223553
first_word_right: 0.8343313373253493
```

仅webtext

```
word right: 0.8544328071987018
sentence right: 0.4231536926147705
first_word_right: 0.8403193612774451
```

webtext+sinanews

```
word right: 0.8753798495353297
sentence right: 0.4530938123752495
first_word_right: 0.852295409181636
```

可以发现字准确率变化不大,但是句准确率上升明显,且使用了webtext数据的都能正确将 阳光开朗大南海输出为 阳光开朗大男孩,数据库中内容类别的增加确实可以提高模型的泛化能力,进而提高准确率

句首字的概率统计

即对于viterbi搜索的第一个字,将其概率初始化为每个字作为句首字出现的概率,下列使用的超参均为上文中得到的最优的超参

- 二元

```
word right: 0.8529871662487093
sentence right: 0.3992015968063872
first_word_right: 0.844311377245509
```

- 三元

```
word right: 0.9055022864729311
sentence right: 0.5508982035928144
first_word_right: 0.8622754491017964
```

结果为在使用相同的参数情况下,二元的结果有所提升,三元的结果有所下降,提升是容易理解的,因为我们提升了句首字的正确率, 我认为三元正确率的下降可能与更改逻辑以后超参不适配有关

使用大模型的**API**进行尝试

- gpt-3.5-turbo-1106

```
word right: 0.5335984869555543
sentence right: 0.23904382470119523
first_word_right: 0.5089820359281437
```

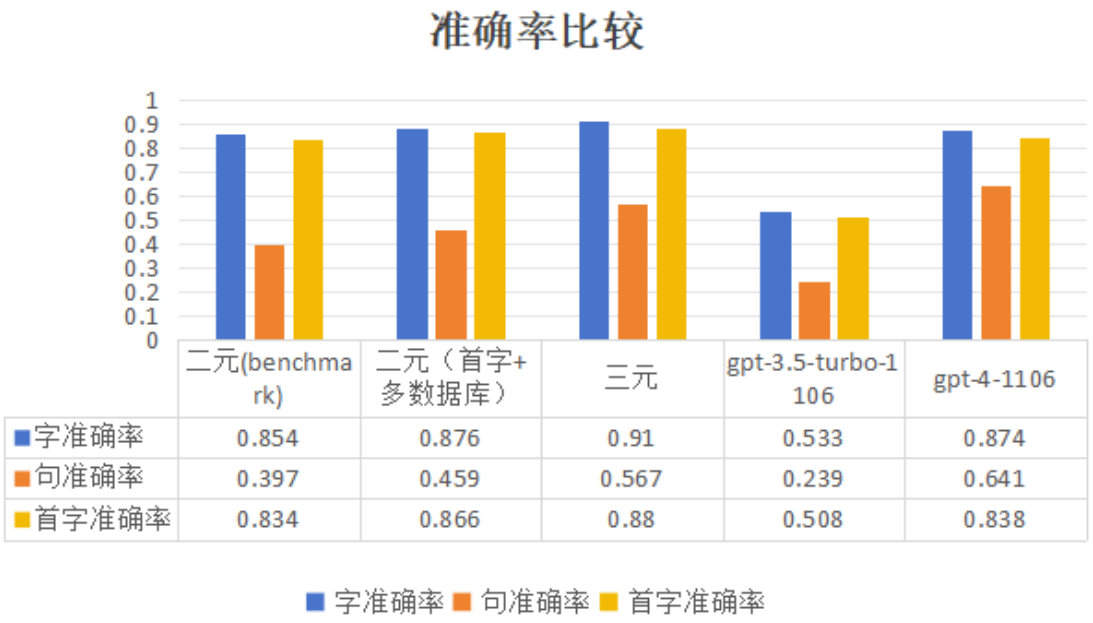
可以看到效果并不见得好，也就是直接使用预训练和大规模tune和对齐的模型可能并不见得能有很好的拼音翻译能力,那么是否可以finetune以后再尝试呢?

- gpt-4-1106

```
word right: 0.8742732343653159
sentence right: 0.6414342629482072
first_word_right: 0.8383233532934131
```

虽然字准确率较低但是句准确率很高,可能gpt-4如果知道某句含义就能准确回答,如果不知道就会出现整句错误的情况

几种方法的正确率比较



	二元(benchmark)	二元（首字+多数据库）	三元	gpt-3.5-turbo-1106	gpt-4-1106
字准确率	0.854	0.876	0.91	0.533	0.874
句准确率	0.397	0.459	0.567	0.239	0.641
首字准确率	0.834	0.866	0.88	0.508	0.838

建议和感受

感觉实现一个输入法还挺有成就感的,但在实际应用中感觉我们似乎也不会输入一个特别长的句子,所以不需要太高元的语法模型, 建议保留此作业,以及希望能在课上听到关于输入法模型中 置信度的讲解,我看了马老师的论文,似乎对于公式中的CF一项也没有过多 的解释,感觉这一项系数可能还挺有意义的,我在实现三元模型时做的一点公式改动就是从这个角度出发的