BoPinyin - 让字模型认出诗句

人工智能导论第一次作业 / 计75班 赵成钢 2017011362

特色

- [x] 二元字模型
- [x] 三元字模型
- [x] 多音字识别
- [x] 引入新的动态规划方法让三元字模型最多考虑7个字
- [x] 分别考虑词内连接语义和词间连接语义
- [x] 计算结果语句同时考虑断句
- [x] 扩展语料库/词库
- [x] OOP风格
- [x] 高性能

依赖

- Python 3
- jieba分词
- pypinyin中文注音

基本使用

- 配置文件为json格式，具体样例如下

```
{
    "data": [
        "data/arts/2016-02.txt",
        "data/arts/2016-04.txt",
        "data/arts/2016-05.txt",
        "data/arts/2016-06.txt",
```

```
            "data/arts/2016-07.txt",
            "data/arts/2016-08.txt",
            "data/arts/2016-09.txt",
            "data/arts/2016-10.txt",
            "data/arts/2016-11.txt"
        ],
        "word": "data/jbc_big.txt",
        "dic": "data/dic.txt",
        "model":
"model/g6_debug_jieba_duoyin_dic_word_break_All.model"
}
```

    - 其中data为训练需要的语料，格式同助教提供的语料文件
    - word为词典文件，dic为拼音对应汉字的表格
    - model为最后输出的训练文件
- 主程序的入口的pinyin.py，有四个参数
    - `--train`参数为指定训练，参数内容为训练配置文件的路径
    - `--test`参数为测试，参数内容为配置文件的路径
    - `--input`参数和`--output`参数相对应，为在测试模式下的输入的拼音文件位置和期望的输出结果位置，如果这两个参数都不写的话最后将会在启动一个Shell
    - 下面提供几个样例
        - 训练：`python3 pinyin.py --train data/config.json`
        - 在Shell中测试：`python3 pinyin.py --test data/config.json`
        - 输入输出都在文件中测试：`python3 pinyin.py --test data/config.json --input input.txt --output output.txt`
    - 如果助教您想直接测试的话(输入输出均为UTF-8格式)
        - 直接输入`python3 pinyin.py --test data/config.json`就好了，将会启动一个类似Shell的东西，每输入一行输出一行结果
        - 语料库加载需要20秒左右，请耐心等待

本地环境

- macOS 10.14.4

- Python 3.7.1
- jieba 0.39
- pypinyin 0.35.1

效果

下面在描述算法前先展示几个比较好的结果

- Case 1: 复杂逻辑语义
  - Input: san yue mo qing hua da xue cheng li le yi ge tian wen xi
  - Output: /三月末/清华大学/成立/了/一个/天文系
- Case 2: 诗句识别
  - Input: wo men dou zhi dao gou li guo jia sheng si yi qi yin huo fu bi qu zhi
  - Output: /我们/都/知道/苟利国家生死以/岂因祸福避趋之
- Case 3: 长词识别
  - Input: liang shan yi zu zi zhi zhou fei chang mei li
  - Output: /凉山彝族自治州/非常/美丽
- Case 4: 新词判断
  - Input: wo he hua lai shi tan xiao feng sheng
  - Output: /我/和/华莱士/谈笑风生
- Case 5: 超长句识别
  - Input: xi jin ping zuo tian dui e luo si jin xing le guo shi fang wen pu jing fei chang gao xing xi jin ping qian li tiao tiao guo lai
  - Output: /习近平/昨天/对/俄罗斯/进行/了/国事访问/普京/非常高兴/习近平/千里迢迢/过来

算法描述

2-Gram

- 我首先使用了最朴素的2-Gram算法，具体分析在上一个字是A的情况下B出现的概率，$P(B|A) = \frac{count(AB)}{count(A)}$，其中$count(str)$为$str$在语料中出现的次数

- 特别的我们需要注意，在$count(A)=0$时，整个式子的值为零
- 由于语料库非常庞大，我们需要对$P(B|A)$进行预先计算，我把这项看成一个矩阵，直接把一个二维数组以二进制的形式打包到一个文件里，实现了训练和测试的分离，这里文件约为$200\ \mathrm{MB}$，里面很多0，压缩后仅有$8\ \mathrm{MB}$左右
- 于是我们可以用动态规划来求解这个问题了
  - 设$f[i][j]$是进行到第i个的拼音，且最后一个字为j这样的概率
  - 那么我们有$f[i][j]=max\{f[i-1][k]*P(j|i)\}$
- 于是出现了下面的问题
  - Input: qu jiao you
  - Output: 区交由
- 我们发现这样的算法过于"短见"，只看到了两个字，连是不是一个词都不会考虑，而且词和词之间和词内也分不清，很容易出现一个字被前后两个字利用成词的情况，那么我们下面考虑下如何简单的断句
- 最简单的断句无非是第一个词的最后一个字和第二个词的第一个字之间没有词间的那么必然的联系，那么第二个词的第一个字如何确定呢?
  - 我们之间用第一个字出现的频率来决定，引入$freq_ch(A)=\frac{count(A)}{length(data)}$来描述一个字出现的频率
  - 进一步我们引入参数α和β，把动态规划改写成$f[i][j]=max\{f[i-1][j]*(\alpha*P(j|i)+\beta*freq_ch(j))\}$
- 这样的话通过简单的调整参数，算法的输出可以稳定的在下面两种情况下变化，令人哭笑不得
  - Input: wo shi ni ba ba
  - Output 1: 我是你爸爸
  - Output 2: 我市是爸爸
  - 原因很简单，算法根本不知道什么情况需要断句，什么情况下是一个词
- 不过我们先解决一个更重要的问题

多音字识别

- 汉语博大精深，多音字就是一个鲜明的体现，在最初的版本中，有一组数据是这样的

    - Input: jiao wai
    - Output: 校外
    - Answer: 郊外
- 可见我们需要引入多音字，这里我采用了pypinyin对语料进行标注，pypinyin会根据语境给出正确的读音，效果一般但也足矣应付大多数的情况
- 在重新训练之后，输入法成功识别了这样的情况

条件概率

- 下一步，我想识别更长的词语和句子，但是在上面的模型中我发现随着长度的变化，最终句子的概率衰减的速度实在是非常快，回到最原始的式子$P(B|A) = \frac{count(AB)}{count(A)}$，我发现其实有两个重要的条件，一个是上一个字是A，下一个是A后面跟的拼音其实我已经知道了，于是我把公式改成$P(B|A) = \frac{count(AB)}{count_with_pinyin(A,pinyin_B)}$，举一个简单的例子，在拼音pin yin下，我计算「拼」这个字后面跟「音」这个字的概率，实际上分子应该是「拼音」这个词在语料中出现的次数，而分母应该是「拼yin」这样的次数，而不是「拼」字出现的次数
- 我们考虑一下，这样是为了什么，这样做可以把条件缩得更紧，所生成的句子中会更加成词，也就意味着词可以更长，因为条件越来越紧，当然也注意到这里仅仅是二元，我现在也只是阐述这种可能性，后面会说明具体的做法说明如何利用这种越来越紧致的条件概率
- 单纯地加上这个所谓的优化，其实在大部分情况下情况会变差，因为这样句子会更加成词，比如"我市你爸爸"这种情况的概率更强了，需要极端的参数才能变回正确的结果，这也意味着我们需要作出适应性地优化

断句

- 正如上面所说的，我们一个很重要的问题是不能区分词间和词中的连接，于是我引入了jieba分词，对两种情况分别进行训练
- 由于jieba分词本身就有一个词库，那么我没有对语料库中词间进行继续分析，而是选择了用jieba对语料进行分词，然后只学习$P(B|A)$在词间的情况
- 下面的问题在与判断是断句好，还是成词好，我给出了下面的规则
    - 断句综合概率
      $P(B|A) = c * P_{break}(A) + d * P_{first}(B) + e * P_{learn}(B|A)$

        - 其中的c, d, e均为可调节参数
        - $P_{break}(A) = \frac{count\_word\_last(A)}{count(A)}$，指的是$A$这个字符是词的最后一个字的概率，这里的两个$count$都从词库中统计出来
        - $P_{first}(B) = \frac{count\_word\_first(B)}{count(B)}$，指的是$B$这个字符是词的第一个字的概率
        - $P_{learn}(B|A)$即为在语料库中学习出来的连接情况
    - 成词概率$P(B|A)$是指在所有的词库中学习出来的连接情况
- 最后经过随缘调参，我得到了更好的结果，毕竟分别考虑了这两种情况，这时有些三字词和四字成语已经可以识别出来了，比如
    - Input: qian li tiao tiao
    - Output: 千里迢迢
    - Ouput before: 千里调调
- 有些断句的情况也还不错，比如我、你、和、的等等一些虚词、人称等等连接都还不错

看的更远

- 这样还是挺没意思的，我想输入一个「岂因祸福避趋之」，输入法还是识别不出来，词库里命名有这个词，但是终究还是太长了，二元根本识别不了
- 于是我把上面的条件概率缩得更加紧致了一些，考虑当前字是所计算的词的第几个字，把$f[i][j]$扩展到了$f[i][j][k]$，即当前是第i个音节，最后一个字是j，而且j是上一个词的第k个字，这样做的好处是，完全把成词和断句分成了两种情况保存，而且计算概率中的条件更加紧致了
- 对于具体的学习有什么区别的，我们发现词间的连接一定是两个字，但是词本身会很长，我们需要做的是修改对词典的学习，于是我把成词概率多加了一维$P(B|A, i)$表示的是在A是词的第i个字的情况下，下一个字是B的概率，具体的计算还是通过统计出现的次数，这里不再赘述
- 而断句的情况呢，我也加入了这点，以防止输入法输出一个不完整的话，也就是把$P_{break}(A)$改成了$P_{break}(A, i)$就是在A作为第i个字结束这个词的概率
- 代码非常麻烦，不过还是写了出来，于是我得到了下面的结果
    - Input: qi yin huo fu bi qu zhi
    - Output: 起因祸福避趋之
    - 还是错了一个字，虽然词到了后面条件越来越紧，但是前面因为二元模

型本身的限制，在看「因」的时候我只能知道前面是「起」

3-Gram

- 好了，没别的办法了写三元吧，一个技巧是把上面的所有的A看成是两个字的组合，其实和二元没有任何的区别
- 而且一个好处是，我这种区分了词间和词内的做法，在语料库中学习的一定是二元的模型，局限就在与词间需要多远的模型
- 最后这样做的效果是，几乎所有的词都可以准确的判断出来，甚至是诗句
  - Input: gou li guo jia sheng si yi qi yin huo fu bi qu zhi
  - Output: 苟利国家生死以/岂因祸福避趋之
  - 完美

其他做的

由于做这个输入法的周期比较长，有一些优化具体到达的效果已经忘记了，下面列举一下，可以和代码进行对照

- 成词奖励：
  - 曾经出现的一个问题是算出来的词虽然语义没有问题，但是实在太生僻了，不如多断几次句，这样我给了一个高频词成词奖励

$$Reward(word) = \begin{cases} 0.01, \ word\ not\ in\ dictionary \\ \min\{\frac{count(word)}{total} * l^{2*length(word)}, 2.0\} \end{cases}$$

  - 意思是低频词或者没出现过的词给一个很低的衰减，高频词给一个奖励，长度越长越生僻，但是是准确的，而一般非常常见的短词，他们自己的频率系数就很大，这样可以避免生僻的短词
  - 其中l是一个参数，需要调节，我最后取的30
- 断句衰减
  - 有的时候语料中有些奇怪的句子连接，这样导致的结果是算法不停断句获得高概率，而正常说话的过程中其实断句的次数没有那么多，于是断一次句就可以乘一个衰减系数q来避免这种奇怪的情况，这里我取的是0.98
- 句子完整性

    - 往往用户的输入是一整句话，这样不仅仅在DP转移的时候需要考虑断句的情况，其实在句子的最后一个本身就是一个断句
    - 于是可以特别判断一下在最后一个音节的时候断句的概率
- 新词识别
    - 有的时候jieba分词并不准确，并不知道是词还是要断句，比如一些奇怪的人名本来不应该分词，但是jieba也把他分开了，于是我们可以把学习到词连接加一部分权重到词间的连接
    - 也就是$P'(B|A,i) = \alpha * P(B|A,i) + \beta * P_{learned}(B|A)$，这里$\alpha$取$0.98$，$\beta$取$0.02$

性能优化

- 全都写完之后，我发现实在是太慢了，算一个10个拼音的句子需要大概1分钟...
- 输出一下看看他算了什么，发现都是没有必要的情况，而且概率非常低，于是我就加了一个阈值来控制，直接把非常低的情况去掉
- 然后把DP改成了类似BFS的做法，用已经算好的f来更新后面的f，这样也迭代更加精确
- 我把阈值设置成$1e-5$之后速度非常快，在我的Intel Core i7 7700HQ上几乎所有的都是在1秒以内可以算出结果，但是对于长句子来说因为衰减的太快了，所以这个阈值可能太大了，所有的结果全都扔掉了，于是可以我设置了一个每三步把阈值乘0.3的操作，基本上正常逻辑的输入都能出结果

还需要做的

- 其实还能写的有很多，但是我实在写不动了... 因为代码已经很多了，再按照下面的思路来几乎要重构了，而且本身我也付出和思考了很多，我也需要把精力多留在正统的学习上，其实我很多的思路也只是trick的存在
- 模型压缩：如果要运行起来我能达到的效果的话，大概需要$200\ \mathrm{MB}$的文件，其实也就是那个词间的学习到的数据，但是其实里面很多全都是0... 我用压缩软件直接压缩之后，其实也只有不到$10\ \mathrm{MB}$
- 词二元、三元模型：私以为我的上述做法某些情况下逼近了词模型，由于精力有限没有再花时间写词模型

- 自动调整参数：最后的模型一共有7个参数，现在的情况都是我凭感觉调出来的，后面进一步的优化也许是做一个模拟退火之类的简单调参程序
- 不好的例子
    - Input: qing hua da xue cheng li le yi ge xin yuan xi
    - Output: /清华大学/成立/了/一个/心愿/系
    - 不足的地方其实比较明显，输入法对词性的全句的语义的了解还是过于短见，最重要的原因还是无法判断词性
    - Input: ding xin ran
    - Output: /定心/燃
    - 「丁欣然」是我女朋友的名字，显然它没有遇到过这个词就是认不出来，还是可以把原因归结于它不知道什么是姓氏的概念，不知道词性，如果我把丁欣然加到词库应该就好了
    - Input: bian cheng shi xian yi ge shu ru fa
    - Output: /变成/实现/一个/输入法
    - 仍然是词性的问题，这个也可以归因于「程/实」这样的连接输入法没学习过

小结

```
Pinyin: ren gong zhi neng dao lun bo da jing shen wo xu yao hao hao xue xi
Result: /人工智能/导论/博大精深/我/需要/好好学习
```