拼音输入法实验报告

计12 2021010724 曾宪伟

0.概要

实现了基于字的二元模型和三元模型的拼音输入法，尝试了使用 Jelinek-mercer 插值和 Interpolated Kneser-Ney 平滑两种平滑算法对概率分布进行调整
使用方法： 在linux终端，先 `pip install -r requirements.txt` 安装依赖,然后在src文件夹，使用 `python3 -m main` 运行，可选参数有 `-m [使用二元（Binary）或三元（Ternary）模型（默认）]`， `-i [输入文件路径]`， `-o [输出文件路径]`， `-a [标准答案文件路径]`

1.算法简介

1.原理

隐马尔可夫模型是统计模型，它用来描述一个含有隐含未知参数的马尔可夫过程。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析(维基百科)
对于拼音输入法这个案例，可观察的参数序列就是输入的拼音，隐藏的参数就是拼音序列对应的汉字序列。
对于HMM，需要定义三个模型：
1.初始概率向量π
2.状态转移概率矩阵A,$A(i,j)$表示从隐藏状态i转移到j的概率
3.发射概率矩阵B，$B(i,j)$表示从隐藏状态i转移到可观测状态j的概率
还需要定义两个假设：
1.**齐次马尔可夫性假设：** 马尔可夫链在任一时刻t的状态只与前一个时刻t-1的状态有关
2.**观测独立性假设：** 任一时刻的观测只与该时刻的状态有关，与其它观测和状态无关。
根据以上定义，原本一个句子$w_1,w_2,...,w_n$出现的概率

$$P(w_1,w_2,...,w_n)=P(w_1)\cdot P(w_2|w_1)\cdot P(w_3|w_1,w_2)\cdots P(w_n|w_1,...,w_{n-1})$$

可以改写为

$$P(w_1,w_2,...,w_n)=P(w_1)\cdot\prod_{i=2}^{n}P(w_i|w_{i-1})$$

又因为我们可以由频率估计概率，即$P(w_i|w_{i-1})\approx\frac{\#(w_{i-1},w_i)}{\#w_{i-1}}$，$\#(w_{i-1}w_i)$即二元汉字组$w_{i-1}w_i$出现的频率。
因此，需要统计每个字出现的频数，所有汉字二元组出现的频数。

2.数据格式化

基于新浪新闻进行训练。首先在ReadData.py中，提取每一则新闻的'html'和'title'得到句子集合，通过pypinyin库函数lazy_pinyin，对每个句子进行注音并将结果保存。对于句子中的非中文字符，对应的拼音使用特殊符号进行占位标注。然后统计所有数据中，每个汉字在某个音标下的出现次数以及所有汉字二元组的出现次数，并将结果保存。

3.算法

使用viterbi算法找出概率最大路径。具体而言，设$w_{i,j}$为第i位汉字的第j个可能的选择，$W_{i,j}$为结尾是$w_{i,j}$的概率最大的前缀，则

$$P(W_{i,j})=max_k(P(W_{i-1,k})\cdot P(w_{i,j}|W_{i-1,k}))$$

每确定一个$W_{i,j}$，就记录这条前缀的w_{i-1}。最后会剩下m条备选路径（m为最后一位汉字的可选择数）再从m条路径中挑选出概率最大的作为最终结果，倒推出这条路径上的所有节点，即为答案。

- **优化细节**
  1.在计算概率的过程中，为了防止浮点精度的丢失导致计算机无法比较，需要对概率取自然对数。
  2.为了使模型不至于在出现了汉字w_i后只局限于预测语料库中出现过的所有w_iw_{i+1}，也为了能够对0概率取对数，需要使用**平滑算法**对模型做适度的松弛。最简单的Laplace法则只是简单地将频数加一，并未考虑和已有数据的权重关系。故采用另一种比较容易实现且效果较好的Jelinek-mercer插值算法:

$$P(w_i|w_{i-1})=\alpha\cdot\frac{\#(w_{i-1}w_i)}{\#w_{i-1}}+(1-\alpha)\frac{\#w_{i-1}}{ratio}$$

其本质是将字二元模型与一元模型做混合，效果是适度减少了出现频率较高的二元组在预测中的概率，提升了出现频率很低甚至为0的二元组在预测中的概率，使整个概率分布更加平滑。

4.拓展尝试

除了基本的二元模型和线性插值平滑算法以外，还尝试了实现三元模型以及实现Interpolated Kneser-Ney平滑算法

- **三元模型**
  三元模型需要额外记录汉字三元组的出现次数。
  在三元模型中，对下一个汉字出现概率的预测依赖于前两个汉字。某个句子$w_1, w_2, \cdots, w_n$出现的概率

$$P(w_1 w_2 \cdots w_n) = P(w_1 w_2) \cdot \prod_{i=3}^{n} P(w_i | w_{i-1} w_{i-2})$$

使用Jelinek-mercer插值算法时，修正预测概率

$$\begin{aligned} P(w_i | w_{i-1} w_{i-2}) &= \beta \frac{\#(w_{i-2} w_{i-1} w_i)}{\#(w_{i-2} w_{i-1})} + (1-\beta) P(w_i | w_{i-1}) \\ &= \beta \frac{\#(w_{i-2} w_{i-1} w_i)}{\#(w_{i-2} w_{i-1})} + (1-\beta)(\alpha \cdot \frac{\#(w_{i-1} w_i)}{\#w_{i-1}} + (1-\alpha)\frac{\#w_{i-1}}{ratio}) \end{aligned}$$

因为语料库规模的限制，三元组的频率已经较低，所以需要适当地混合低维度的一元、二元模型。

同样使用viterbi算法，设$w_{i,j}$为第i位汉字的第j个可能的选择，$W_{i,j,k}$为当前结束字符在第i位，末尾字符为$w_{i,j}$，倒数第二个字符为$w_{i-1,k}$的概率最大的前缀，则

$$P(W_{i,j,k}) = max_k(P(W_{i-1,k,s}) \cdot P(w_{i,j} | W_{i-1,k,s}))$$

每确定一个$W_{i,j,k}$，就记录$w_{i-1,k}$。在选出概率最大路径后，往前倒推出所有节点。

- **Interpolated Kneser-Ney平滑算法**
  在插值的时候，考虑到以下情况：w_i是一个生僻字，w_i只有在前面为w_{i-1}时才会出现，但是$w_{i-1}w_i$是一个二元组且在语料库中出现的次数较多。而相比之下，w_j是一个常见字，可以跟在很多字后面，但刚好这些二元组在语料库中出现频次较少。此时为了提高w_j概率，需要对模型进行修正，具体做法为

$$P^*(w_i | w_{i-1}) = \lambda \cdot \frac{\#(w_{i-1} w_i)}{\#(w_{i-1})} + (1-\lambda) P_{cont}(w_i)$$

  其中$P_{cont}(w_i) = \frac{N(*w_i)}{N(**)}$，*为通配符。也就是低阶项取 *结尾是$w_i$的二元组的种类数量* 与 *所有二元组的种类数量* 的比值

5.实验结果

二元模型
在使用二元模型预测时，对于不同的α和$ratio$，正确率如下：

α	逐字正确率	整句正确率
0.999	64.48%	6.38%
0.9999	78.39%	23.95%
0.99999	84.83%	38.72%
0.999999	85.73%	42.31%
0.9999999	85.4%	42.91%
0.99999999	85.36%	42.51%

（以上ratio均为10000）

$ratio$	逐字正确率	整句正确率
1000	85.73%	42.31%
10000	85.4%	42.91%
100000	85.36%	42.51%

（以上α均为0.9999999）
综合考虑，取$\alpha = 0.9999999$，$ratio = 10000$

三元模型
在使用三元模型预测时，对于不同的$\alpha, \beta, ratio$，结果如下：

α	逐字正确率	整句正确率
0.999	89.36%	52.49%
0.9999	92.28%	65.46%
0.99999	93.1%	70.25%
0.999999	93.25%	69.66%

（以上$\beta, ratio$分别为0.95，10000）

β	逐字正确率	整句正确率
0.91	93.33%	70.25%
0.93	93.23%	70.45%
0.95	93.1%	70.25%
0.97	93.14%	70.05%
0.99	92.89%	69.06%

（以上$\alpha, ratio$分别为0.99999,10000）

$ratio$	逐字正确率	整句正确率
1000	92.24%	65.26%
10000	93.23%	70.45%
100000	93.33%	69.86%

（以上α, β分别为0.99999,0.93）

可以发现，从二元模型到三元模型，预测的正确率有了大幅度的提升。而参数的细微调整，只会带来一些小的优化。

Interpolated Kneser-Ney平滑

以上的模型均基于线性插值，那么若使用Interpolated Kneser-Ney平滑，效果会更好吗？

在三元模型使用线性插值的当前最优参数下，使用Interpolated Kneser-Ney平滑，逐句正确率为**68.86%**，逐字正确率为**93.25%**，劣于线性插值。而后又尝试了不同的参数以及二元模型，结果均为正确率差于线性插值。推测原因与语料库有关。语料库来源于新闻，数据分布较均匀，不会出现常见词语频次少而偏僻词语频次较大的极端情况。

6.案例分析

1.

三元模型总体表现好于二元模型，例如：

对zhong guo chen wen ying dui mao yi mo ca，三元模型给出“中国沉稳应对贸易摩擦”，二元模型给出“中国**陈文**应对贸易摩擦”，对于ta niu tou xiang si zhou kan le kan，二元模型因为前向的局限性会给出“他扭头**像**四周看了看”，但三元模型能给出正确答案“他扭头向四周看了看”

2.

但是，也存在二元模型能预测正确而三元模型反而预测错误的例子，例如：

对ni ru he kan dai e wu zhan zheng，二元模型给出“你如何看待俄乌战争”，而三元模型给出“你如何看待俄**午**战争”；对于er ci yuan wen hua kai shi fa zhan,二元模型给出“二次元文化开始发展”，而三元模型给出“二次元文化开始发**站**”。推测原因，是因为一些三元组的耦合比较紧，比如“始发**站**”“甲**午**战争”是一些固定短语，导致预测时倾向认为结果是这些固定搭配的子串。

3.

当然也存在一些两种模型都不能预测对的句子，比如都把“勇敢猫猫不怕困难”预测成了“用干毛毛不怕困难”，以及“他是我的母亲”这种结果。对于前一种情况，需要更多的偏向于日常生活和网络用语的语料库，对于后一种情况，可能需要引入更完善的模型。

7.结论

在本次实验中，我学习了对海量数据的处理和组织方法，以及如何从中提取有用的信息。同时我还体验了定义模型、评价函数、调参的过程。训练集对模型的预测效果会产生相当的影响，评价函数的好坏直接影响结果的质量，参数的微调带来更加细致的优化。

当前的改进方向：

1.优化句首和句尾的预测，在句首句尾，对常出现在句首句尾的汉字的概率赋予更多的权重

2.进行意群划分，减小一个完整意义的短语对后面的汉字的粘连性

3.加入更多的训练集如贴吧发言、聊天记录等。