人工智能导论——拼音输入法实验报告

计 93 王哲凡 2019011200

2021 年 4 月 20 日

目录

1 算法介绍 **2**
1.1 数据统计 2
1.2 字二元模型 2
1.3 求模型最优解 3
1.4 字三元模型 4

2 实验结果 **5**
2.1 准确率展示 5
2.2 正确案例 6
2.3 错误案例 6
2.4 案例分析 7

3 运行方法与环境 **7**
3.1 环境要求 7
3.2 使用方法 8

4 总结 **8**

1 算法介绍

本次实验我主要采用了字的二元模型和三元模型算法。

对于每个潜在的汉字选项，我们给出了其对应的评价函数。

1.1 数据统计

首先遍历给定的 sina 新闻数据集，提取其中的 title 和 html 部分，将每个部分作为一个潜在的学习语料，此部分在 `src/datareader.py` 中的 `DataReader` 类中实现。

然后我们通过 `pypinyin` 库对语料进行预处理的拼音标注，此处采用了库中的 `lazy_pinyin()` 方法，主要用以处理语料中多音字的具体读音情况。

结合通过上面方法标注的拼音，我们遍历所有 sina 语料，统计了其中各个汉字字的出现次数（记录在 `dataset/frequency-1.json` 下），以及相邻二字组成的不同二元组出现次数（记录在 `dataset/frequency-2.json` 下），对于字三元模型，则进一步统计了相邻三个汉字构成的三元组的出现次数（记录在 `dataset/frequency-3.json` 下），这部分处理均由 `src/preprocess.py` 中的函数完成。

除此之外，我还统计了单拼音对应汉字的字典 `dataset/word_table.json` 和单汉字对应拼音的字典 `dataset/pinyin_table.py`。

`dataset` 文件夹下 `sentences-XX.json` 为 sina 语料的预处理中间文件，并未被模型使用，而 `word_frequency-X.json` 则为通过 `jieba` 库分词后的词频率统计，目前暂未使用。

1.2 字二元模型

对给定的一个拼音序列 $\{p_1, p_2, \cdots, p_n\}$，其中 p_i 为其中第 i 个汉字的拼音，我们希望得到的是在给定这个拼音序列条件下，最高概率的汉字序列（即句子），即：

$$W_0 = \underset{w_1, w_2, \cdots, w_n}{\arg\max} \; P(w_1 w_2 \cdots w_n | p_1 p_2 \cdots p_n)$$

根据贝叶斯公式：

$$P(w_1 w_2 \cdots w_n | p_1 p_2 \cdots p_n) = \frac{P(p_1 p_2 \cdots p_n | w_1 w_2 \cdots w_n) \cdot P(w_1 w_2 \cdots w_n)}{P(p_1 p_2 \cdots p_n)}$$

> **贝叶斯公式**：
>
> $$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

由于 $\{p_1, \cdots, p_n\}$ 序列确定，故 $P(p_1 p_2 \cdots p_n)$ 为常数，而 $P(p_1 p_2 \cdots p_n | w_1 w_2 \cdots w_n)$ 可认为是 1（通过将字拆分成对应读音的分字可保证其为 1），因此我们只须关注 $P(w_1 w_2 \cdots w_n) = P(W)$。

通过条件概率可化简为：

$$
\begin{aligned}
W_0 &= \underset{w_1,w_2,\cdots,w_n}{\arg\max}\ P(W) \\
&= \underset{w_1,w_2,\cdots,w_n}{\arg\max}\ P(w_1w_2\cdots w_n) \\
&= \underset{w_1,w_2,\cdots,w_n}{\arg\max}\ P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_n|w_1w_2\cdots w_{n-1}) \\
&\approx \underset{w_1,w_2,\cdots,w_n}{\arg\max}\ P(w_1)P(w_2|w_1)P(w_3|w_2)\cdots P(w_n|w_{n-1}) \\
&= \prod_{i=1}^{n} P(w_i|w_{i-1})
\end{aligned}
$$

其中 w_i 表示第 i 个汉字的预测，$P(w_i|w_1w_2\cdots w_{i-1})$ 表示前 $i-1$ 个字分别确定为 $w_1,\cdots,w_{i-1}$ 时，第 i 个字为 w_i 的概率，$P(w_i|w_{i-1})$ 表示前一个字为 w_{i-1} 时，后一个字为 w_i 的概率（在给定拼音条件下），此处默认 $P(w_1|w_0)=P(w_1)$ 即第一个拼音对应 w_1 的概率。

又根据条件概率公式：

$$P(w_i|w_{i-1})=\frac{P(w_{i-1}w_i)}{P(w_{i-1})}$$

其中 $P(w_{i-1}w_i)$ 即表示 w_{i-1} 与 w_i 相邻出现的概率，$P(w_{i-1})$ 表示 w_{i-1} 出现的概率。

根据大数定律，我们可以用语料中的统计频率来逼近估计，即：

$$P(w_i|w_{i-1})\approx\frac{\text{count}(w_{i-1}w_i)}{\text{count}(w_{i-1})}=P^*(w_i|w_{i-1})$$

其中 $\text{count}(w_{i-1}w_i),\text{count}(w_{i-1})$ 分别表示语料中 $w_{i-1}w_i$ 和 w_i 出现的频次。

而考虑到部分 $P(w_{i-1}w_i)$ 较低的情况，可能导致在语料库中 $\text{count}(w_{i-1}w_i)$ 退化为 0，因此通过 laplace 平滑可将上式改写为：

$$P(w_i|w_{i-1})\approx\alpha P^*(w_i|w_{i-1})+(1-\alpha)P^*(w_i),\alpha\approx 1$$

其中 $P(w_i)$ 的估计 $P^*(w_i)$ 为：

$$P^*(w_i)=\frac{\text{count}(w_i)}{r}$$

> 模型中统一取 $r=100000$。

1.3 求模型最优解

模型文件为 `src/models.py`，其中构造函数实现了读入语料数据，`forward()` 函数实现了最优预测。

模型中的下标为 $0,\cdots,n-1$，与下面的规定略有不同。

设 $\text{dp}_{i,w}$ 表示前 i 个汉字，第 i 个为 w 的概率对数最大值，$\text{last}_{i,w}$ 则表示取得最大值情况下选择的第 $i-1$ 个汉字。

初始值为：

$$\text{dp}_{1,w} = \log P(w), w \in \text{Word}(p_1)$$

其中 $\text{Word}(p)$ 表示拼音 p 所对应的汉字集合。

容易得到转移方程：

$$\begin{cases} \text{dp}_{i+1,w} = \max\limits_{w' \in \text{Word}(p_i)} \left(\text{dp}_{i,w'} + \log P(w|w')\right) \\ \text{last}_{i+1,w} = \arg\max\limits_{w' \in \text{Word}(p_i)} \left(\text{dp}_{i,w'} + \log P(w|w')\right) \end{cases}$$

通过动态规划容易求解得到完整的 dp 和 last 结果，再通过 last 数组倒推即可得到完整的预测汉字序列。

> 此处的 P 均为 1.2 模型中给出的语料统计估计值。

1.4 字三元模型

通过单个汉字预测下一个汉字的二元模型有较大的的局限性，考虑通过前两个汉字来分析下一个汉字，即：

$$\begin{aligned} W_0 &= \arg\max_{w_1,w_2,\cdots,w_n} P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_n|w_1w_2\cdots w_{n-1}) \\ &\approx \arg\max_{w_1,w_2,\cdots,w_n} P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_n|w_{n-2}w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_{i-2}w_{i-1}) \end{aligned}$$

同样需要考虑退化情况，处理为：

$$\begin{aligned} P(w_i|w_{i-2}w_{i-1}) &\approx \beta P^*(w_i|w_{i-2}w_{i-1}) + (1-\beta)P(w_i|w_{i-1}) && \beta \approx 1 \\ &\approx \beta P^*(w_i|w_{i-2}w_{i-1}) + (1-\beta)\left[\alpha P^*(w_i|w_{i-1}) + (1-\alpha)P^*(w_i)\right] && \alpha \approx 1 \end{aligned}$$

其中：

$$P^*(w_i|w_{i-2}w_{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-2}w_{i-1}w_i)}{\text{count}(w_{i-2}w_{i-1})} & \text{count}(w_{i-2}w_{i-1}) > 0 \\ 0 & \text{otherwises.} \end{cases}$$

模型求解可类比 1.3 中的二元情况，只须将 dp 和 last 拓展一维即可。

2 实验结果

下面测试使用的语料均为提供的 sina 新闻语料，测试输入文件为 `data/input.txt`，结果文件为 `data/output.txt`。

2.1 准确率展示

字二元模型的准确率如下：

α	整句正确率	逐字正确率
0.9	1.80%	55.01%
0.99	7.06%	64.44%
0.999	23.64%	77.54%
0.9999	36.28%	83.17%
0.99999	**39.08%**	**83.83%**
0.999999	38.91%	83.53%
0.9999999	38.58%	83.50%

表 1: 不同 α 值字二元模型表现

字三元模型的准确率如下：

参数 α	参数 β	整句正确率	逐字正确率
0.9999	0.999	64.20%	91.21%
	0.99	65.02%	91.38%
	0.95	**65.51%**	91.51%
	0.9	65.18%	91.74%
0.99999	0.999	63.71%	91.24%
	0.99	64.53%	91.41%
	0.95	65.18%	**91.76%**
	0.9	65.18%	91.74%
0.999999	0.999	63.38%	91.23%
	0.99	64.20%	91.36%
	0.95	64.86%	91.72%
	0.9	64.36%	91.67%

表 2: 不同 α, β 值字三元模型表现

可见字三元模型相对字二元模型，无论是整句正确率还是逐字正确率都有了较明显的提升，特别是整句正确率。

二元模型中，α 的值设置在 0.999 以下时，效果较差，特别是对于整个句子的解析基本都不到位；而在 0.9999 以上，效果基本接近，基本差别不大。

三元模型中，取了二元模型表现较好的几个 α 值进行测试，可以发现 β 值对于正确率的影响较小，基本在 $\beta = 0.95$ 左右时效果最佳。

2.2 正确案例

下面选取一些拼写正确的案例来证明输入法的效果，均为字三元模型的案例。

- `qing hua da xue shi shi jie yi liu da xue`
  清华大学是世界一流大学
- `ting che zuo ai feng lin wan`
  停车坐爱枫林晚
- `xun xun mi mi leng leng qing qing qi qi can can qi qi`
  寻寻觅觅冷冷清清凄凄惨惨戚戚
- `mei ge si nian yi ci de ao yun hui jiu yao zhao kai le`
  每隔四年一次的奥运会就要召开了
- `wei ji bai ke shi yi ge wang luo bai ke quan shu xiang mu`
  维基百科是一个网络百科全书项目
- `yi ge zi ren wei qian li bu fan de ren tong guo jian ku zhuo jue de nu li`
  一个自认为潜力不凡的人通过艰苦卓绝的努力
- `zhong guo pin kun di qu shi xian wang luo fu wu quan fu gai`
  中国贫困地区实现网络服务全覆盖
- `ben ci pu cha huo dong you zhu yu bang zhu tong xue men zou chu xin li wu qu`
  本次普查活动有助于帮助同学们走出心理误区
- `xiao chu kong ju de zui hao ban fa jiu shi mian dui kong ju`
  消除恐惧的最好办法就是面对恐惧
- `duo qu xin shi dai zhong guo te se she hui zhu yi wei da sheng li`
  夺取新时代中国特色社会主义伟大胜利

2.3 错误案例

下面选取一些拼写错误的案例，均为字三元模型的案例。

- `yi zhi ke ai de da huang gou`
  输出：一只可爱的大**皇沟**
  正解：一只可爱的大**黄狗**

- `pin yin zhi jian yong kong ge ge kai`
  输出：**品音**之间用空格隔开
  正解：**拼音**之间用空格隔开

- `kai tong jin jin si shi ba xiao shi xi fen er shi jiu wan`
  输出：开通仅仅四十八小时**细分**二十九万
  正解：开通仅仅四十八小时**吸粉**二十九万

- `ni jia wo de wei xin`
  输出：你**枷**我的微信
  正解：你**加**我的微信

- `hua wei dui dai cheng xu yuan ru he`
  输出：**化为**对待程序员如何
  正解：**华为**对待程序员如何

2.4 案例分析

根据正确案例，可以看到，对于长难句，乃至一些简单的古文，输入法都能给出准确的回答。

但错误案例中，也有一些较为简单的句子未能给出一个正确的回答，甚至给出的回答是不太合理的。

这一方面体现的是语料库本身可能有一些侧重点的问题，比如**黄狗**判断成**皇沟**可能与语料库中较少有“大黄狗”等表达有关。

另一方面，输入法对于较长的上下文缺乏判断能力，比如**加**我的微信被判断为了**枷**，以及**华为**判断为**化为**，这是字三元模型算法本身的一个弊端。

总结而言，模型的弊端主要分为**算法本身的局限**和**语料库的偏向性**。对于后者，一些更大更全面的数据集或语料库可能可以较好地解决；对于前者，则可能需要加入词多元模型或者 NLP 模型，使得模型可以更深入理解句子本身。

3 运行方法与环境

3.1 环境要求

除用于输入输出分离的源文件 `process.py` 位于 `data` 文件夹下外，所有的 `.py` 源文件均位于 `src` 文件夹下。

Python 环境建议为 Python 3.8，主要用到的 Python 库如下（可参考 `requirements.txt` 文件）：

- `tqdm` 4.59.0
- `pypinyin` 0.41.0
- `jieba` 0.42.1

其余所用库均为 Python 自带库。

3.2 使用方法

模型在命令行中使用：

- `usage: python -m src.main [-h] [--model MODEL] [--input INPUT] [--output OUTPUT] [--nocheck] [--answer ANSWER]`
- `-m，--model` 指定模型名称，可选 `BinaryModel` 和 `TernaryModel`，默认为 `TernaryModel`。
- `-i，--input` 指定输入文件，默认为 `data/input.txt`。
- `-o，--output` 指定输出文件，默认为 `data/test.txt`。
- `-a，--answer` 指定答案文件，默认为 `data/output.txt`。
- `-n，--nocheck` 指定是否比对答案，默认为 0，即比对。

4 总结

本次输入法实验是我第一次做一个有实践意义的人工智能实验，这次实验让我对于人工智能领域中的一些概念，比如评价函数、预测模型、参数设置等有了更深入的了解。

通过实验，我获得了以下结论：

- 更多元的模型会有更好的效果，但同时也会带来更大的性能与负载压力。
- 不同的参数设置在不同模型中取得的效果可能会有较大差别，对于参数的调整是十分有必要的。
- 语料库对于模型的建立十分重要，不同的语料库在相同的测试下可能导致较大的结果差异。

除此之外，通过实验与数据分析，我也思考了模型的几个可能的改进方向：

- 尝试利用分词来增加词的二元乃至三元转移，可能可以对于一些出现频率较低的词汇有更好的效果。
- 增加更多的语料库，比如增加 bilibili、知乎等平台的语料，有助于加强对于一些新兴词汇的解析。
- 考虑引入神经网络等结构，通过一些 NLP 的模型来加入语义理解等。