人工智能导论: 拼音输入法

计 13 班 2021010761 程思翔

1 文件结构

1.1 程序目录结构

```
├── README.txt
├── corpus/
│   ├── sina_news_gbk/
│   │   └── ...
│   └── webtext2019zh/
│       └── web_text_zh_train.json
├── data/
│   ├── input.txt
│   ├── output_v0.txt
│   ├── output_v1.txt
│   ├── output_v2.txt
│   ├── output.txt
│   └── std_output.txt
└── src/
    ├── frequency-list/
    │   ├── merged/
    │   │   ├── corpus_merge.py
    │   │   └── merged_3gram.json
    │   ├── sina/
    │   │   ├── sina_2gram.json
    │   │   ├── sina_2gram.py
    │   │   ├── sina_3gram.json
    │   │   └── sina_3gram.py
    │   └── web_text/
    │       ├── web_text_2gram.json
    │       ├── web_text_2gram.py
    │       ├── web_text_3gram.json
    │       └── web_text_3gram.py
    ├── pinyin-dict/
    │   ├── pinyin_dict.json
```

```
    │   └── pinyin_dict.py
    ├── pinyin_2gram.py
    ├── pinyin_3gram_fix.py
    ├── pinyin_3gram.py
    ├── pinyin.py
    └── utils.py
```

2 算法思路 - 基于字的二元模型

2.1 问题定义

给定一串拼音, 实质上我们需要求得

$$\max P(S) = \max \prod_{i=1}^{n} P(\omega_i \mid \omega_{i-1})$$

所对应的句子. 也即求

$$\min \left(-\sum_{i=1}^{n} \log P(\omega_i \mid \omega_{i-1}) \right)$$

所对应的句子.

2.2 算法思路

- 从语料库构建二元词频表. 如果不进行任何优化操作, 对句首字的判断仅能依靠于其在语料库中出现的概率, 这是不合理的, 因为汉字出现在句首的概率并不等于其出现在句中的概率.
- 因此在每个句子前加上一个 `#` 号 (假设每个句子都以 `#` 开始), 这样原来的句首汉字与句中汉字就有了相同的地位. 在构建词频表时, 只需同时统计形如 `#啊`, `#我`, `#今` 的二元词组即可.
- 使用 `cur_state` 列表记录当前读入拼音对应的句子前缀及它们出现的概率. `cur_state` 初始化为 `[('#', 1)]`, 表示句首字为 `#`, 出现的概率为 `1`.
- 顺序遍历给出的拼音, 我们考虑其对应的位于一二级汉字表的所有汉字. 对于每个可行的汉字 `current`, 我们考虑其紧跟着 `cur_state` 中句子的最后一个汉字出现的概率 `p(current|last)`, 并记录下使得这个概率最大的汉字 `last` (相当于一种剪枝).
- 考虑到平滑处理, 我们定义参数 `ALPHA`, `BETA` 加权考虑汉字的可能出现情况:
    - `ALPHA` 用于平衡汉字的单独出现概率;
    - `BETA` 用于平衡汉字出现在词语中的概率;

- $$P(current \mid last) = \alpha \cdot P(current) + \beta \cdot \frac{P(last - current)}{P(last)}$$
  $$= \alpha \cdot \frac{cnt(current)}{cnt(All)} + \beta \cdot \frac{cnt(last - current)}{cnt(last)}$$
  - 其中 `cnt(last)` 表示语料库中 `last` 出现的次数, `cnt(last-current)` 表示 `last` 和 `current` 一起出现的次数;
  - 经过测试, 选取 `ALPHA=0.001`, `BETA=0.999`.
- 若 `cur_state` 列表中原来记录了长为 `k` 的句子出现的概率, 经过一轮操作后我们得到了长为 `k + 1` 的句子出现的概率. 考虑到不同句子的组合数量级很大, 我们取概率前 10 大的句子作为下一轮操作的 `cur_state`. 最后只需输出 `cur_state` 中概率最大的元素对应的句子 (删去句首 `#`) 即可.

3 实验环境

- **编程语言**: `python 3.9` (只需要创建一个版本为 `3.9` 的虚拟环境即可)
- **操作系统**: `MacOS Ventura 13.0`

4 语料库选择与数据处理

4.1 建立拼音 - 汉字表

- 读入拼音汉字表及一二级汉字表;
- 建立并存储 `dict`: **{拼音:[汉字]}**, 其中汉字列表只保留一二级汉字.

4.2 建立词频表

- 选择了**新浪新闻** `sina_news_gbk` 目录下的语料.
- 使用正则表达式 `[^\u4e00-\u9fa5]` 匹配所有非汉字的字符, 并替换为空格;
- 在每个句子前加上一个 `#` 号, 在构建词频表时, 同时统计形如 `#啊`, `#我`, `#今` 的且出现在拼音 - 汉字表中的二元词组;
- 每碰到两个字, 将它们在词频表中的 `count` 加 1.

5 样例测试 - 准确率

在 `src` 目录下使用如下命令进行测试:

```
python pinyin_2gram.py ../data/input.txt ../data/output_v0.txt
```

可得初始版本 `v0` 的准确率 (还是非常不错的!):

```
单字正确率为：83.4766%
整句正确率为：40.3194%
```

6 效果展示

6.1 优秀样例

- ji qi xue xi shi dang xia fei chang huo re de ji shu
- **机器学习是当下非常火热的技术**
- ta de ren sheng gui ji jiu shi yi ge tan xin suan fa
- 他的人生轨迹就是一个贪心算法
- jing ji jian she he wen hua jian she tu chu le shi ba da jing shen de zhong yao xing
- **经济建设和文化建设突出了十八大精神的重要性**
- zhe zhong qing kuang hui dui hang mu da ji qun de ren yuan zao cheng ying xiang
- **这种情况会对航母打击群的人员造成影响**
- ou zhou ren xi huan ba gu lao de jiao tang zan mei wei ning gu de yin yue
- **欧洲人喜欢把古老的教堂赞美为凝固的音乐**

6.2 较差样例

- 互联网上开放的中文语料库有哪些
- 互联网上开放的**重温预**料库有哪些
- 中国是人民民主专政的社会主义国家
- 中国**市**人民民主**转正**的社会主义国家
- 清仓大甩卖
- **青藏**大甩卖
- 疫情面前中塞两国再次携手共克时艰
- **一情**面前中塞两国再次携手**工课时间**
- 认真地数过星星
- 认真**的**数**国性行**

6.3 样例分析

- **多音字**: 一些常见字具有不常见的读音, 在程序输出时会因为这些常见字本身出现的概率较大, 而对识别产生干扰.
    - 如: 中文语料库 - **重温预**料库
- **二元语法**: 每个字的选择只能依据其前后两个字, 无法考虑到句子的整体语义.
    - 如: 人民民主专政 - 人民民主**转正**
    - 如: 共克时艰 - **工课时间**

- **语料库**: 由于使用了新浪新闻语料库, 且语料时间均在 2016 年, 输入法倾向于将拼音识别为新闻中的高频率词, 且对于一些 “新” 词的识别能力很差.
    - 如: 疫情 - 一情

7 拓展: 字的三元模型

7.1 算法思路

- 取构造好的二元词频表, 统计包含这些二元词组的三元词组出现的次数. 因为数量级过大, 我们只保留出现次数前 5×10^6 的三元词组.
- 在每个句子前加上一个 `##` 号 (即假设每个句子都以 `##` 开始), 这样可以统计形如 `##今`, `#今天` 的三元词组, 消除了句首字因为其特殊位置而产生的统计误差.
- 使用 `cur_state` 列表记录当前读入拼音对应的句子前缀及它们出现的概率. `cur_state` 初始化为 `[('##', 1)]`, 表示句首字为 `##`, 出现的概率为 `1`.
- 顺序遍历给出的拼音, 我们考虑其对应的位于一二级汉字表的所有汉字. 对于每个可行的汉字 `current`, 我们考虑其紧跟着 `cur_state` 中句子的最后一个汉字出现的概率 `p(current|last)`, 并记录下使得这个概率最大的汉字 `last` (相当于一种剪枝).
- 考虑到平滑处理, 我们定义参数 `ALPHA`, `BETA` 加权考虑汉字的可能出现情况:
    - `ALPHA` 用于平衡汉字的单独出现概率;
    - `BETA` 用于平衡汉字出现在词语中的概率;
    - `GAMMA` 用于平衡汉字出现在三元词组中的概率;
    - $$\begin{aligned} & P(cur \mid first-last) \\ =& \alpha \cdot P(cur)+\beta \cdot \frac{P(last-cur)}{P(last)}+\gamma \frac{P(first-last-cur)}{P(first-last)} \\ =& \alpha \cdot \frac{cnt(cur)}{cnt(All)}+\beta \cdot \frac{cnt(last-cur)}{cnt(last)}+\gamma \frac{cnt(first-last-cur)}{cnt(first-last)} \end{aligned}$$
    - 其中 `cnt(last)` 表示语料库中 `last` 出现的次数, `cnt(last-cur)` 表示 `last` 和 `cur` 一起出现的次数, `cnt(first-last-cur)` 表示 `first`, `last` 和 `cur` 一起出现的次数.

7.2 样例测试

在 `src` 目录下使用如下命令进行测试:

```
python pinyin_3gram.py ../data/input.txt ../data/output_v1.txt
```

可得改进版本 `v1` 的准确率:

```
单字正确率为：88.6724%
整句正确率为：56.0878%
```

注意到整句准确率明显升高, 这是因为三元模型相比于二元模型, 使得句子内容判断更具有连贯性.

8 拓展: 语料库选择与性能

8.1 webtext2019zh

- 源自 **GitHub** 项目[brightmart/nlp-chinese-corpus](), 含 410 万个问答 (过滤后数据 3.7G, 数据跨度 2015-2016 年), 本次作业选用了其中前 6×10^5 行数据进行训练.
- 使用模型为字的三元模型.
- 在 `src` 目录下使用如下命令进行测试:

```
python pinyin_3gram_fix.py ../data/input.txt ../data/output_v2.txt
```

可得改进版本 `v2` 的准确率:

```
单字正确率为：91.1366%
整句正确率为：61.8762%
```

8.2 混合语料库

- 基于 `sina_news_gbk` 和 `webtext2019zh`, 考虑将其词频表加权得到新的词频表. 经过测试, 设定其权值比为 `1:4`.
- 在 `src` 目录下使用如下命令进行测试:

```
python pinyin.py ../data/input.txt ../data/output.txt
```

可得改进版本 `v3` 的准确率:

```
单字正确率为：92.5692%
整句正确率为：67.4651%
```

- 这是因为多个语料库加权融合, 可以实现汉字频率的取长补短, 综合各语料库的优势, 使得词频表更具有普遍性.

9 拓展: 参数选择与性能

9.1 二元情况 - 以 pinyin_2gram.py 为例

α	0.001	0.01	0.05	0.1	0.2
单字正确率	83.4766%	83.3912%	83.3429%	83.3620%	83.1328%
整句正确率	40.3194%	40.3194%	39.3214%	39.3214%	36.9261%

最终选取 `ALPHA=0.001`, `BETA=0.999`.

9.2 三元情况 - 以 pinyin.py 为例

出于 `9.1` 中的考虑, 我们选取 `ALPHA=0.001`, 考虑 `BETA` 的取值.

β	0.001	0.01	0.1	0.2	0.3
单字正确率	90.6781%	91.8052%	92.4928%	92.5692%	92.3018%
整句正确率	63.4731%	65.4691%	67.0659%	67.4651%	65.6687%

最终选取 `ALPHA=0.001`, `BETA=0.2`, `GAMMA=0.799`.

10 总结与备注

10.1 总结

- 语料库应选取大而广, 避免选择的局限性.
- 三元模型的准确率最高, 但是构造和运行时间也相对较长.
- 尝试使用词的模型进一步提高准确率.

10.2 备注

- `pinyin_2gram.py` 为新浪语料库 - 二元模型算法.
- `pinyin_3gram.py` 为新浪语料库 - 三元模型算法.
- `pinyin_3gram_fix.py` 为社区问答语料库 - 三元模型算法.
- `pinyin.py` 为多语料库加权 - 三元模型算法, 实现效果最好.