拼音输入法实验报告

《人工智能导论》第一次作业

任自厚

2022 年 3 月 18 日

1 拼音输入算法

本次实验讨论的拼音输入算法，是接收合法且正确的全拼输入、输出单一的预测汉字字符串的离线算法。要实现拼音输入法，需要在给定一段拼音序列后，将每个拼音转换为汉字，使之连贯起来语法正确、语义通顺。

1.1 基于字的算法概述

在基于字的算法中，每个字被视为独立单元进行计算，而不考虑其与前后某些字可能组成的通用词等。基于字的算法优点在于模型简单、无需进行分词处理，同时又可以凭借庞大的语料库达到较优的预测效果。其劣势则在于对某些常用词的预测没有基于词的算法理想。

拼音 $\rightarrow$ 汉字的映射为一对多的关系（不考虑多音字，或将多音字视为同形的不同汉字），故可以将拼音输入算法转化成如下问题：**给定候选汉字集 $w_{i,j}(1 \le i \le n)$，选择汉字序列 $w_1 w_2 \dots w_n$，使得该序列得分 s 最高**。

我们用每个字 $w_{i,j}$ 出现在 i 位置的概率计算序列得分：

$$s = P(w_1 w_2 \dots w_n) = \prod_{i=1}^{n} P(w_i | w_1 w_2 \dots w_{i-1})$$

为了便于计算，实际评价指标 d 为概率的负对数：

$$d = -\log s = \sum_{i=1}^{n} \log \frac{1}{P(w_i | w_1 w_2 \dots w_{i-1})}$$

此处 d 越小越好，故问题进一步转化为最短路问题：**给定一个节点为字符的图（形如图 1），求起点（虚构字符 $\wedge$）到终点（虚构字符 \$）的最短路，并打印路径**。

因此，只要对任意相邻节点给出良好的距离定义 d，即可求解汉字序列。这是一个动态规划算法。

考虑到汉语语法中，一个词的字数一般不超过 4 个，故基于字的算法也通常只考虑当前预测字 w_i 前方 $1 \sim 3$ 个字的影响。这样可以在保障准确率的前提下，降低网络复杂度、减少模型大小、提升计算速度。在本次实验中，我分别实现了基于字的二元、三元、四元模型。这些模型的差别仅在于节点距离的计算。算法推导与模型评价如下。

1.2 基于字的二元模型的算法

在字的二元模型下，$P(w_i | w_1 w_2 \dots w_{i-1}) = P(w_i | w_{i-1})$。特别的，$P(w_1) = P(w_1 | \wedge)$，$P(\$) = P(\$ | w_n)$。

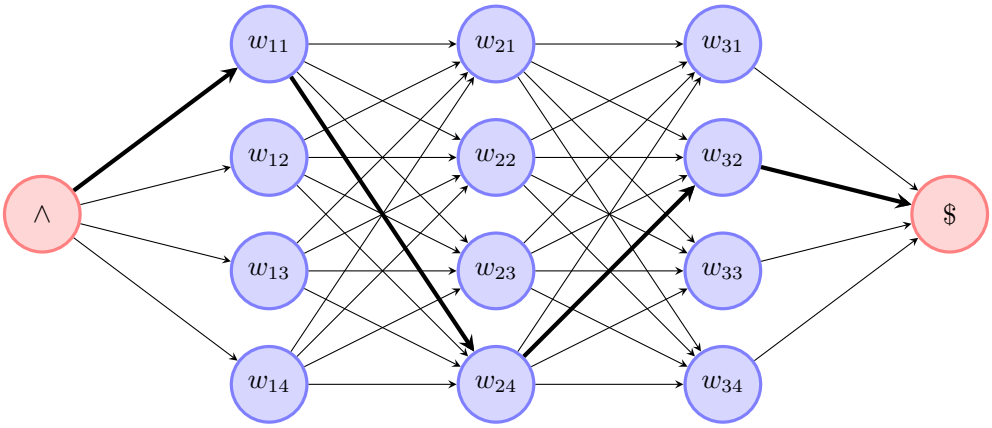

图 1: 字符为节点的决策图

Input: nodes
Output: sentence
`/* initialize layers with only beginning character */`
$layers \leftarrow [[(\text{ '}\wedge\text{ '}, 0, None)]]$;
`/* dp from layer to layer, always choose currently minimum distance path */`
for *candidates* ***in*** *nodes* **do**
  $layer \leftarrow []$;
  for *char* ***in*** *candidates* **do**
    $dists \leftarrow [pred[1] + d(w, pred[0])$ `for` *pred* `in` $layers[-1]]$;
    $layer$.append((*char*, min(*dists*), argmin(*dists*)));
  end
  $layers$.append($layer$);
end
$dists \leftarrow [pred[1] + d(\text{ '\$ '}, pred[0])$ `for` *pred* `in` $layers[-1]]$;
$layers$.append([(*char*, min(*dists*), argmin(*dists*))]) ; `// only ending character`
`/* trace backwards to get the final sentence */`
$sentence \leftarrow$ '';
$p\_index \leftarrow 0$;
for *layer* ***in*** *layers* **do**
  $sentence$.addToBegin($layer[p\_index][0]$);
  $p\_index \leftarrow layer[p\_index][2]$;
end

Algorithm 1: DP to Sentence

根据以上公式，可知构建统计数据时，需要记录每个汉字 w 紧跟在另一个汉字 v 后面的频数 $c(vw)$、w 出现在句首、句尾的频数 $c(\wedge w), c(w\$)$。并由此计算 $P(w|v) = c(vw)/c(v)$。句首、句尾处理类似。

特别的，我们需要考虑某一汉字组合从未出现在语料库中的情形，故引入平滑处理：

$$P_2(w_i) = \lambda \frac{c(w_{i-1}w_i)}{c(w_{i-1})} + (1-\lambda)\frac{c(w_i)}{c_{\text{all}}}$$

所以实际上，二元模型是“一元”与“二元”的结合。对于从未在语料库中的生僻字，则将其赋予一个极小的非零概率 ϵ。

1.3 基于字的三元、四元模型的算法

仿照二元模型，我们可以构建三元、四元模型：用单字频率、双字频率、三字频率（、四字频率）的加权和作为 $P(w_i|w_1 \dots w_{i-1})$。具体公式如下：

$$P_3(w_i) = \mu \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})} + \lambda \frac{c(w_{i-1}w_i)}{c(w_{i-1})} + (1-\lambda-\mu)\frac{c(w_i)}{c_{\text{all}}}$$

$$P_4(w_i) = \nu \frac{c(w_{i-3}w_{i-2}w_{i-1}w_i)}{c(w_{i-3}w_{i-2}w_{i-1})} + \mu \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})} + \lambda \frac{c(w_{i-1}w_i)}{c(w_{i-1})} + (1-\lambda-\mu-\nu)\frac{c(w_i)}{c_{\text{all}}}$$

当前缀长度不够时，这些模型会退化为二元模型。

2 拼音输入算法的 Python 实现

2.1 实验环境与数据

实验所用 Python 版本为 Python 3.9.5。为了更好展现统计及预测进度，程序使用了第三方库 tqdm 4.63.0。不过，该库非必要依赖，在导入失败时程序仍会正常执行，只不过缺少进度条显示与倒计时。

在运行四元模型的统计阶段时，由于需要维护一个庞大的词频表，程序对内存有一定要求。在我的电脑（16G 内存）上，运行至该阶段时，物理内存已经无法满足需求，大量的虚存访问与页交换严重降低了程序运行速度，以致我不得不在服务器上进行实验。

基本原始数据为课程助教提供的新浪新闻语料库（sina_news）和含有 500 条句子的测试集。在下文，我们也会在维基百科语料库（wikipedia）和百科问答语料库（baike_qa）两个数据集上进行词频统计，并仍在原测试集上给出测试效果。

2.2 Python 实现思路

本程序主要包括两部分：语料库词频统计与拼音预测。为了易于扩展，语料库处理和拼音汉字网络计算均使用子类继承的方式。具体代码结构如下：

语料库处理类定义在 preprocess.py 中，基类为 DataProcessor，实现了读取语料库、提取特定条目中连续汉字文本的功能。该类还包含两个抽象方法 _analyse() 和 _calc_prob()，分别用于统计词频、计算频率。其有三个派生类 BiGramProcessor、TriGramProcessor、QuadGramProcessor，分别为二元、三元、四元语言模型进行词频统计。词频统计时，我放弃了所有频数为 1 的样本，因为这些样本大概率来自于词语连接部分，对模型精度无大幅提升、又会占据较大存储空间。

拼音汉字网络计算类定义在 graph.py 中，基类为 CharacterGraph，实现了上文所述动态规划算法。该算法会调用抽象方法 _dist() 计算节点距离。其有三个派生类 BiGramGraph、TriGramGraph、QuadGramGraph，分别加载二元、三元、四元语言模型进行计算与预测。

为了便于量化模型效果，还实现了评估脚本 validate.py，可统计字正确率和句正确率。

所有模型（词频统计）以 json 格式保存在 src/stat 目录下。为了压缩空间，所有频率值只保留 5 位有效数字。src/stat 目录下还包括 mapping.json 文件，定义了拼音到汉字的映射关系。实验用到的所有模型文件均可从清华云盘下载。

程序入口为 run.py，该脚本接收 4 个命令行参数，使用方法如下：

```
python3 src/run.py \
  --task <stat|predict|val> \
  --model <bigram|trigram|quadgram> \
  --input /path/to/input/file \
  --output /path/to/output/file
```

其中，task 参数默认为 predict，model 参数默认为 bigram。若要执行模型构建（词频统计），task 选择 stat，input 传入语料库路径（支持文件夹或单文件），output 传入模型文件输出路径；若要运行输入法，task 选择 predict，input 传入拼音文件路径，output 传入输出文件路径，程序会自动加载 src/stat 目录下同名模型；若要统计准确率，task 选择 val，input 和 output 分别传入标准输出与程序输出文件路径（此时 model 参数无效）。

3 算法评价

3.1 参数评价

这里主要讨论上文出现的 λ, μ, ν 三个重要模型参数的取值。

我们已经通过预实验确定，词频表未命中时采用的微小非零词频分别为 $\epsilon_1 = 10^{-4}, \epsilon_2 = 10^{-7}$, $\epsilon_3 = 10^{-6}, \epsilon_4 = 10^{-5}$。这里 ϵ_i 表示查找的词为 i 元词。这部分实验内容不再赘述。

对于二元语法模型，参数只有一个平滑系数 λ。实验尝试了若干个典型的 λ 值，并在准确率极大值附近加细试探，结果如表 1。从表中可以看出，在 $\lambda = 0$ 时，二元退化为一元，节点之间互不影响，每个字都预测为字频最大者，效果不佳。另一方面，在 $\lambda \to 1$ 时，模型准确率也有所下降，这是由于语料库覆盖不全面，部分二元字词未能统计到。在 $\lambda = 0.993$ 时，模型同时具有较好的字句准确率。

λ	0	0.8	0.85	0.9	0.95	0.99	**0.993**	0.999	1
character	0.5104	0.8411	0.8392	0.8432	0.8434	**0.8446**	**0.8446**	0.8425	0.8421
sentence	0.0060	0.3960	0.3920	0.4020	0.4060	0.4180	**0.4200**	0.4120	0.4120

表 1: 二元语法模型参数对准确率的影响

对于三元语法模型，存在两个参数 λ, μ，分别控制二元、三元词频在距离贡献中的权重。实验在令 $1 - \lambda - \mu = 0.01$ 的前提下，尝试了若干个典型的参数取值，结果如表 2。可以看到，在 $\mu = 0$ 时，三元退化为二元，准确率与二元模型相近。另一方面，在 $\mu \to 1$ 时，二元因素完全不做贡献，模型表现也较差。这可能是因为汉语中仍以二元词语居多，而仅靠三元词频统计很难较好的匹配出二元词组，有害于模型精度。在 $\lambda = 0.39, \mu = 0.60$ 时，模型同时具有较好的字句准确率。

(λ, μ)	(0.99, 0)	(0.69, 0.3)	**(0.39, 0.6)**	(0.09, 0.9)	(0, 0.99)
character	0.8398	0.8972	**0.9080**	0.9017	0.8075
sentence	0.4100	0.5720	**0.6060**	0.5840	0.3120

表 2: 三元语法模型参数对准确率的影响

对于四元语法模型，存在三个参数 λ, μ, ν，分别控制二元、三元、四元词频在距离贡献中的权重。实验在令 $1-\lambda-\mu-\nu = 0.01$ 的前提下，尝试了若干个典型的参数取值，结果如表 3。类似三元模型，四元模型也会在其控制参数 μ 较大时因样本不够全面而有损精度。在 $\lambda = 0.29, \mu = 0.50, \nu = 0.20$ 时，模型同时具有较好的字句准确率。

(λ, μ, ν)	(0.09, 0.4, 0.5)	(0.09, 0, 0.9)	**(0.29, 0.5, 0.2)**	(0.29, 0.2, 0.5)	(0.49, 0, 0.5)
character	0.9059	0.8658	**0.9079**	0.9015	0.8794
sentence	0.5980	0.4960	**0.6060**	0.5860	0.5560

表 3: 四元语法模型参数对准确率的影响

3.2 模型评价

我们从两个角度对模型进行评价：预测准确率和预测成本。

预测准确率包括字准确率（character_accuracy）和句准确率（sentence_accuracy）。

在参数评价一节中，我们已经得到了各个模型较优的参数。表 4 横向比较了各个模型在准确率上的表现，可以看出三元模型相较二元有着较大的提升，而四元模型仅仅与三元持平。

accuracy	BiGram	TriGram	QuadGram
character	0.8446	**0.9080**	0.9079
sentence	0.4200	**0.6060**	**0.6060**

表 4: 各个模型准确率比较

为了更直观的体现各模型优劣，表 5 给出了两个预测样例。从第一个样例中，可以看出随着模型元数增加，模型对较长词语（如“得寸进尺”、“永不满足”）的识别能力有所提升。而第二个样例则说明了，现有模型在多音字处理上表现较差。如果仅看预测语句，“买柜子”和“买车子”都是合法且通顺的语言，但输入拼音为“ju”，不对应前面二者任何一个符合语义的读音。因此若要解决这个问题，就需要引入对多音字的辨析。由于目前的语料库并未提供汉字读音，这一工作暂时无法开展。

预测成本包括构建成本与预测时间，前者又可从时空两个维度进行评价。

前文已经提及，在运行至四元语法模型时，笔记本的 16G 内存已经无法支持程序高效运行；其实在三元模型下，程序对内存的消耗就已经非常显著了。由此可见，更多元的模型对内存空间有更严格的要求。

由于空间消耗不易测量，我们主要从运行时间的角度进行比较。表 6 是各个模型在 sina_news 语料库上的构建时间、加载模型时间以及测试集上的预测耗时。

综合以上各个指标，三元语法模型预测准确率高，且资源消耗在可接受范围内，是在当前数据集下的更优解。

std	贪婪的人总是得寸进尺永不满足	我去给你买一个橘子
BiGram	贪婪的人从事的村金池用不满足	我去给你买一个车子
TriGram	贪婪的人总是得寸进尺用不满足	我去给你买一个柜子
QuadGram	贪婪的人总是得寸进尺永不满足	我去给你买一个车子

表 5: 预测样例

time (seconds)	BiGram	TriGram	QuadGram
stat task (sina_news)	**306.46**	821.40	1411.79
load model (sina_news)	**1.21**	12.12	39.75
predict task (500 sentences)	**3.36**	6.64	8.63

表 6: 各个模型时间消耗

3.3 语料库评价

本节旨在评价不同语料库统计出的词频模型对预测精度的影响。在这一阶段，我们使用三元语法模型，参数设置为 $\lambda = 0.39, \mu = 0.60$。

表 7 展示了不同语料库的规模大小、预测精度。可以看出，新闻语料库 sina_news 表现最好。这可能是因为新闻语言同时具备通俗、规范的性质。相较之下，维基百科语料库 wikipedia 的语言更为专业，术语较多，导致其在主要包括常用语的测试集上表现不佳；而问答语料库 baike_qa 的内容为用户生成，虽贴近生活，但存在语法不规范、语句杂乱的现象，影响了词频统计。

corpus (λ, μ)	sina_news (0.39, 0.60)	wikipedia (0.37, 0.62)	baike_qa (0.49, 0.50)
corpus size (compressed) (MB)	463.9	514.1	624.4
stat file size (MB)	322.5	343.5	325.7
character accuracy	**0.9080**	0.8698	0.8966
sentence accuracy	**0.6060**	0.4580	0.5760

表 7: 不同语料库构建出的模型比较