

# Input Method

Pinyin input method in Python. Homework of Intro. to AI course, 2022 Spring @THU.

[zhaochen20](#) THUCST Class 6

## The Principle

### 隐 Markov 模型 (HMM)

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process — call it X — with unobservable ("hidden") states. As part of the definition, HMM requires that there be an observable process Y whose outcomes are "influenced" by the outcomes of X in a known way. Since X cannot be observed directly, the goal is to learn about X by observing Y. HMM has an additional requirement that the outcome of Y at time  $t = t_0$  may be "influenced" exclusively by the outcome of X at  $t = t_0$  and that the outcomes of X and Y at  $t \leq t_0$  must not affect the outcome of Y at  $t = t_0$ .

对于长度为  $n$  个汉字的中文序列对应的拼音序列  $S = s_1s_2 \dots s_n$ , 需要确定每个单字拼音  $s_i$  对应的中文字符  $w_i$ , 使得中文序列  $w_1w_2 \dots w_n$  最佳。

借助现有的语料库, 我们可以统计出任意中文字串  $W$  与输入串  $S$  的匹配概率  $P(S)$ ; 该值越大, 意味着输入串  $S$  的输入结果更可能是  $W$ 。也即, 最好的输入效果即最大化  $P(w_1w_2 \dots w_n)$ 。

$$P(w_1w_2 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2) \dots P(w_n|w_1w_2 \dots w_{n-1}) \quad (1)$$

因此, 最大化  $P(w_1w_2 \dots w_n)$  也即于最大化:

$$\ln P(w_1w_2 \dots w_n) = \ln P(w_1) + \ln P(w_2|w_1) + \ln P(w_3|w_1w_2) + \dots + \ln P(w_n|w_1w_2 \dots w_{n-1}) \quad (2)$$

$P(w_k|w_1w_2 \dots w_{k-1})$  意味着在前缀  $w_1w_2 \dots w_{k-1}$  确定的情况下,  $w_k$  紧随其后的概率。该概率可以看做从长度为  $k-1$  的状态通过  $w_k$  转移到长度为  $k$  的状态的权重。基于此, 引入  $m$  阶隐 Markov 模型, 即每次状态的转移只与目标状态的前  $m$  (某个常数) 个状态有关, 而与后面的状态和前  $m$  个状态之前的状态无关。于是, 问题简化为最大化:

$$\sum_{i=1}^n \ln P(w_i|w_{i-m} \dots w_{i-1}) \quad (3)$$

暂时忽略下标为负的项。

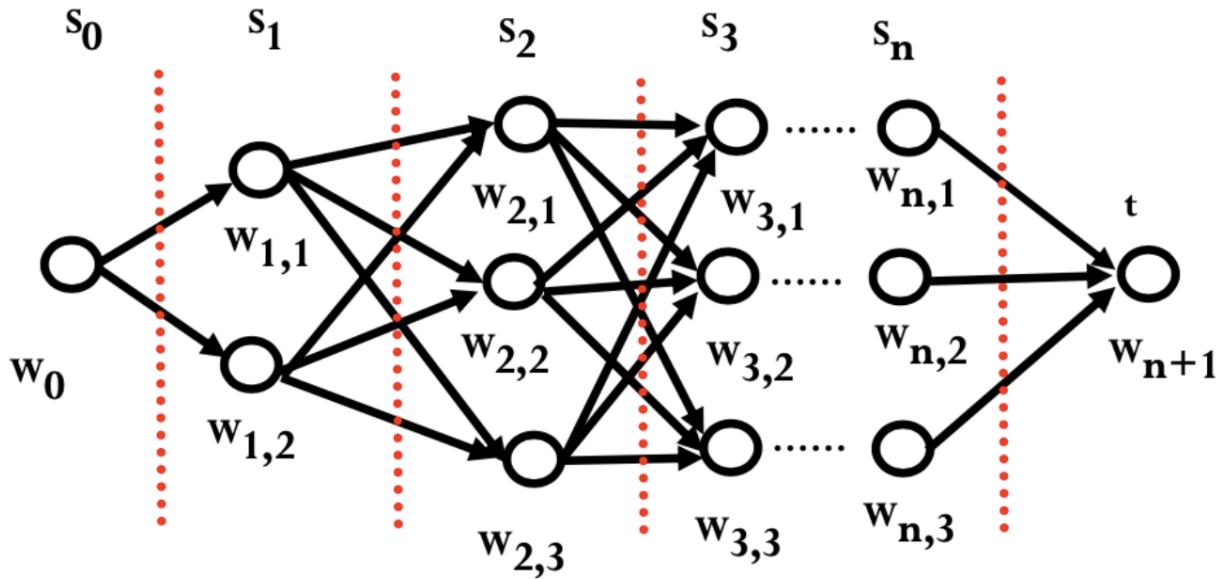
### Viterbi 动态规划算法

上述模型中, 令  $m = 1$ , 即让每次状态转移只与前1个状态转移有关 (称“二元语法”), 则问题转化为最大化:

$$\sum_{i=1}^n \ln P(w_i|w_{i-1}) \quad (4)$$

该问题可进一步转化为利用 Viterbi 算法在一个有向无环图中寻找最长路径的问题。

为了问题更加准确, 我们考虑三元语法, 并对其进行简化。



图中记号  $s_i$  对应各个单字拼音，节点  $w_{i,j}$  为单字拼音  $s_i$  对应的第  $j$  个可选汉字（ $w_0$  为起始符， $w_{n+1}$  为终结符）。对于各个节点  $w_{i,j}$ ，下标中  $i$  值相邻的节点间必有指向  $i$  值增大方向的有向边  $(w_{i,j}, w_{i+1,k})$ ，其一元权值记录为  $P(w_{i+1,k})$ ，其二元权值记为  $P(w_{i+1,k}|w_{i,j})$ ，即在汉字  $w_{i,j}$  出现的条件下， $w_{i+1,k}$  紧随其后出现的概率，其三元权值记录为  $P(w_{i+1,k}|w_{i,j}w_{i-1,x})$ ，其中， $w_{i-1,x}$  是上一步转移到  $w_0 \dots w_{i,j}$  后存储的（最佳）路径信息中的第  $i$  项（第 1 项为  $w_0$ ，是起始符）。经过简单测试，发现完全遍历第  $i$ 、 $i-1$  层的完全三元语法运行速度太慢，为了减少运算时间，采用了这一模型。

## 具体权重

### 一元模型

$$P^*(w_i) = \frac{\text{count}(w_i)}{r} \quad (5)$$

模型中统一取  $r = 10\,000\,000$ 。

### 二元模型

$$P(w_i|w_{i-1}) = \frac{P(w_{i-1}w_i)}{P(w_{i-1})} \approx \frac{\text{count}(w_{i-1}w_i)}{\text{count}(w_{i-1})} = P^*(w_i|w_{i-1}) \quad (6)$$

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。

进一步加权：

$$P(w_i|w_{i-1}) \approx \alpha P^*(w_i|w_{i-1}) + (1 - \alpha)P^*(w_i), \alpha \approx 1 \quad (7)$$

### 三元模型

$$P^*(w_i|w_{i-2}w_{i-1}) = \begin{cases} \frac{\text{count}(w_{i-2}w_{i-1}w_i)}{\text{count}(w_{i-2}w_{i-1})} & \text{count}(w_{i-2}w_{i-1}) > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

进一步加权：

$$P^*(w_i|w_{i-2}w_{i-1}) \approx \beta \frac{\text{count}(w_{i-2}w_{i-1}w_i)}{\text{count}(w_{i-2}w_{i-1})} + (1 - \beta)((\alpha P^*(w_i|w_{i-1}) + (1 - \alpha)P^*(w_i)), \alpha, \beta \approx 1 \quad (9)$$

## 代码实现

```

1 cost(characters: str, coefficient: List[float]) -> float:
2     """
3         Since the three unit cost is determined by three characters, we need three characters
4         to get the cost.
5         """
6
7     x = coefficient[0]
8     y = coefficient[1]
9     model = len(characters)
10    try:
11        if model == 1:
12            try:
13                return math.log(one[f"{characters}"] / 10_000_000)
14            except:
15                return -10000
16        elif model == 2:
17            try:
18                return math.log((1 - x) * one[f"{characters[1]}"] / 10_000_000 + x *
19 (two[f"{characters}"] / one[f"{characters[0]}"]))
20            except:
21                return -10000
22        else:
23            try:
24                return math.log(y * three[characters] / two[characters[0:2]] + (1 - y) *
25 ((1 - x) * one[f"{characters[2]}"] / 10_000_000 + x * (two[f"{characters[1:3]}"] / one[f"
26 {characters[1]}"])))
27            except:
28                return -10000
29    except Exception as e:
30        embed()
31        exit()

```

为了避免生僻字的影响，当出现生僻字时，对其进行惩罚，权重设为 -10000。

## 文件说明

```

1 .
2   └── README.md
3   └── data
4       ├── input.txt
5       └── output.txt
6   └── pic
7       ├── full_set.png
8       ├── half_set.png
9       └── log.png
10  └── src
11      └── complete.sh

```

```
12     └── dictionary.npz
13     └── full_set.npz
14     └── half_set.npz
15     └── pinyin.py
16     └── pipeline.py
17     └── refactor.npz
18     └── refactor_data.py
19     └── requirements.txt
20
21 3 directories, 15 files
```

以上为关键文件，其中 refactor\_data.py 用于将原始 json 文件清洗为纯文本 json 文件，train.py 用于统计一元，二元与三元数据，并将其储存为 npz，pinyin.py 为最后的执行文件。

complete.sh 与 pipeline.py 为调参测试使用的脚本。

refactor.npz 采用原生语料库训练，去除了长度小于 5 的字句。half\_set.npz 采用原生语料库训练，没有去除任何句子。full\_set.npz 加入了百度问答语料库，比原生语料库扩大了三倍。

上述代码由于路径设置问题，不能直接运行，可以直接运行的代码参考如下 [github 链接](#) 与 [清华云盘链接](#)。

## 具体实验

### 数据集与实验环境

数据集为给定的 Sina 新闻数据集，实验环境为 MacOS Monterey for M1，测试部分由于在本地运行压力较大，故而在服务器运行，服务器系统如下：

Distributor ID: Ubuntu

Description: Ubuntu 20.04.1 LTS

Release: 20.04

Codename: focal

使用的库主要有 p\_tqdm, pypinyin, IPython, Numpy, tqdm, Pathlib, Argparse, json, collections，具体可参考  
[./src/requirements.txt](#)

### 数据清洗

- 具体代码为 [./src/refactor\\_data.py](#)
- 考虑到除了句号与逗号之外的符号不影响，因此去除这部分标点符号，然后以逗号和句号将句子拆分为分句
- 为了避免文件过大，去除长度小于 5 的分句

事后想来，这可能严重影响了较短分句的效果

- 将文件格式化为中文字字符串+|+中文字字符串的格式
  - 例如：苟利国家生死以 | 美国的华莱士比你们不知道高到哪里去了 | 没这个能力 |
- 得到的字符串存储在 json 文件中，并存储在 Large 文件夹下
- 将原始字典也进行清洗，采用了拼音指向字符列表的方式，存储在 [./src/dictionary.npz](#)

# 训练

- 具体代码为 `./src/train.py`
- 通过简单的统计，进行训练
- 核心思想是构造了 `Neuron` 类，分别统计某个字符串单元单个文字，二元词组与三元词组的出现次数。
- 利用 `Counter`, `Argparse` 优化读取与字典存储
- 利用 `p_tqdm` 采用多线程加快了训练速度（大致训练花费在服务器上为 4 分钟）
- 利用 `numpy` 将训练结果存储为 `./src/refactor.npz`

# 计算

- 具体代码为 `./src/pinyin.py`
- 运行参考如下

```
1 | python3 pinyin.py -i /Users/zhaochen20/THU_CST/2022_spring/人智导/作业/input-method/测试语  
料/input_2.txt -o ./test.txt -c 1 1
```

- 可通过 `python3 pinyin.py -h` 获取帮助
- 通过数据清洗阶段构造的拼音到字符列表的字典，构造了计算图
- 从计算图最内层出发，基于 Viterbi 动态规划算法展开计算
- 具体算法与权重如上方所示，此处以 `Point` 类辅助实现

# 实验结果

由于字准确率均较高( 高于 60% )，故针对句准确率做具体分析。

`x` 表示二元语法的系数，`y` 表示三元语法的系数，两系数越大则对于语法占比越高（参考上文所写权重部分）

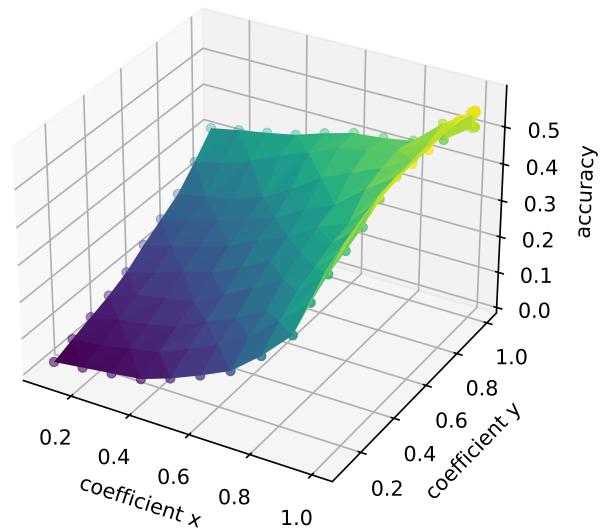
`log.png`：采用原生语料库训练，去除了长度小于 5 的字句。

`half.png`：采用原生语料库训练，没有去除任何句子。

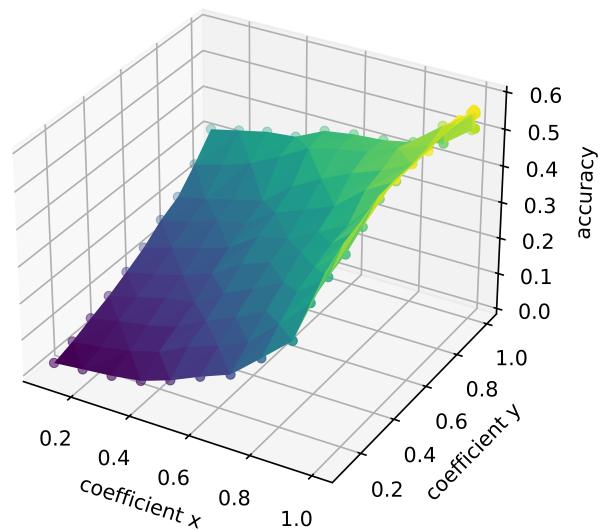
`full_set.npy`：加入了百度问答语料库，比原生语料库扩大了三倍。

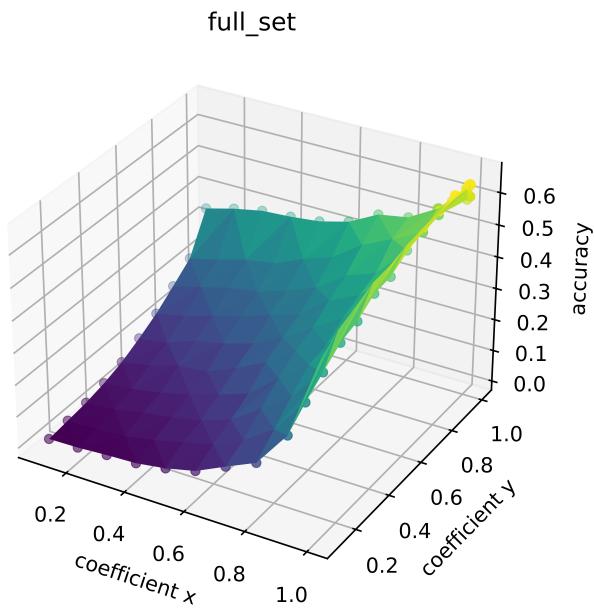
# 训练热力图

log



half\_set





## 准确率分析

- 三幅热力图的共同变化趋势展示，句正确率随着  $x$  与  $y$  的增大而上升，从最低接近全错(此时为完全的单字概率模型)到最高 60+% (采用全训练集与几乎完全的三元语法)
- 句正确率在  $x = 1.0$   $y = 0.9$  时，在全数据集上达到最大，为 64.6 %

## 案例分析

### 正确案例

以下展示一些三元模型正确的模型：

- mei guo de hua lai shi bi ni men bu zhi dao gao dao na li qu le  
美国的华莱士比你们不知道高到哪里去了
- qing hua da xue ji suan ji xi jin nian you you san ge ren tui xue le  
清华大学计算机系今年又有三个人退学了
- mei guo jia ni fu ni ya zhou you shi suo zhu ming de gong li xue xiao  
美国加尼福尼亚州有十所著名的公立大学
- cheng nian ren de sheng huo mei you rong yi er zi  
成年人的生活里没有容易二字
- 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
- wo men dou you guang ming de wei lai  
我们都有光明的未来

### 错误案例

以下展示一些三元模型正确的模型：

- chun feng hua yu le wei yang  
春风化娱乐未央  
春风化雨乐未央

2. bei jing shi shou ge ju ban guo xia ao hui yu dong ao hui de cheng shi  
北京市首个举办过夏奥会于冬奥会的城市  
北京是首个举办过夏奥会于冬奥会的城市
3. da liang chu yu la ji yu qi ta la ji hun he tian mai huo feng shao  
大量厨余垃圾余其他垃圾混合填埋或焚烧  
大量厨余垃圾与其他垃圾混合填埋或焚烧
4. li lao shi xi huan hai kuo tian kong de chang tan  
李老师喜欢海阔天空的长潭  
李老师喜欢海阔天空的长谈
5. nan jing shi chang jiang da qiao  
南京市昌江大桥  
南京市长江大桥
6. sheng hua huan cai  
生化环猜  
生化环材

## 具体分析

---

- 参考正确案例，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，比如长江判断成昌江是因为语料库中有几个关于昌江市的地理信息介绍，而较少有关于长江的表达。
- 另一方面，输入法对于较长的上下文缺乏判断能力，比如厨余垃圾与其他垃圾被判断为了厨余垃圾余其他垃圾，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。

## 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

基于隐 Markov 模型、viterbi 算法，我成功实现了拼音转汉字的任务。除此之外，在评测效果、撰写报告的过程中，我还积累了 Python 中 argparse, numpy 等 package 的[实践经验](#)。