

## 《自然语言处理》实验报告

年级、专业、班级	2018 级计算机科学与技术卓越 01 班	姓名	屈湘钧
实验题目	HMM 模型训练实践		
实验时间	2020/12/7	实验地点	DS3401
实验成绩		实验性质	<input type="checkbox"/> 验证性 <input type="checkbox"/> 设计性 <input type="checkbox"/> 综合性
<p>教师评价：</p> <p><input type="checkbox"/>算法/实验过程正确； <input type="checkbox"/>源程序/实验内容提交 <input type="checkbox"/>程序结构/实验步骤合理；</p> <p><input type="checkbox"/>实验结果正确； <input type="checkbox"/>语法、语义正确； <input type="checkbox"/>报告规范；</p> <p>其他：</p> <p>评价教师签名：</p>			
<p>一、实验目的</p> <p>理解、掌握隐马尔可夫模型，N 元语法等自然语言处理的基本思想、算法，并将其应用于从汉语拼音到汉字的自动转换过程。</p>			
<p>二、实验项目内容</p> <p>(1) 对训练语料及相关资源进行预处理；</p> <p>(2) 通过学习算法，训练 HMM 模型；</p> <p>(3) 使用合理的数据平滑方式，解决 N 元文法中的数据稀疏问题；</p> <p>(4) 利用 HMM 模型和维比特算法，实现从任意拼音到汉字的自动转换，要求程序能对转换中的歧义进行正确识别和处理。</p> <p>(5) 利用给定测试集，评价上述程序的转换准确率。</p>			
<p>三、实验过程或算法（源程序）</p> <p>（一）HMM 模型</p> <p>对于一个随机事件，有一个可以观测到的值序列：<math>O_1O_2...O_T</math></p> <p>该事件的每一个观察到的值 <math>O</math> 都对应一个生成他的状态 <math>S</math>，则其背后存在一个状态序列：<math>q_1q_2...q_T</math></p> <p>假设 1：（马尔科夫假设）每一个状态的值都与其前 <math>n</math> 个状态的值相关</p> <p>假设 2：（不动性假设）状态与具体的时间无关</p> <p>假设 3：（输出独立性假设）输出只与当前状态有关</p> <p>则一个 HMM 模型是一个五元组 <math>(\Omega_s, \Omega_o, A, B, \pi)</math></p>			

其中

$\Omega_S = \{q_1 \dots q_n\}$  状态集合

$\Omega_O = \{v_1 \dots v_t \dots v_n\}$  观察集合

$A = \{a_{ij}\}, a_{ij} = p(q_t = s_j | q_{t-1} = s_i)$  转移概率

$B = \{b_{ik}\}, b_{ik} = p(O_t = v_k | q_t = s_i)$  发射概率

$\pi = \{\pi_i\}, \pi_i = p(q_1 = s_i)$  转移概率

解码问题：对于给定的模型  $(\Omega_S, \Omega_O, A, B, \pi)$  和观察值序列  $O_1 O_2 \dots O_T = v_1 v_2 \dots v_t$ ，求出最大可能性的状态序列  $q_1 q_2 \dots q_T = v_1 v_2 \dots v_t$ 。

## (二) 拼音转汉字算法设计

拼音转汉字即对应 HMM 模型的解码问题。由已知的语料库训练出来汉字到汉字的转移概率和汉字到拼音的发射概率，然后用户输入拼音序列为已知的观察值序列，求大嘴可能性的汉字状态序列。

用一个简单的例子来表示这个识别的过程及原理。

若用户想在计算机得到汉字“我爱中国”，则需要往键盘敲入“wo ai zhong guo”这四个英文字符串。从 HMM 模型出发，“wo ai zhong guo”是观测值序列，如下观测流程图。

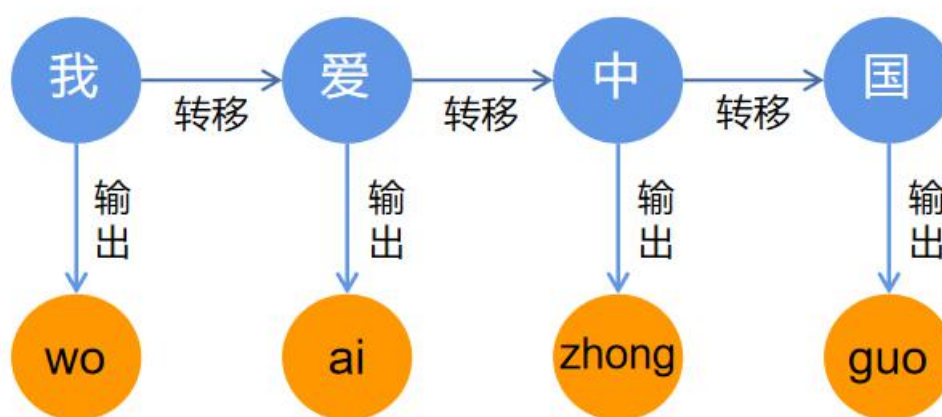


图1 我爱中国汉字拼音输入 HMM 观测过程

图中蓝色圆圈为隐藏的状态，即汉字，橙色圆圈代表可以观测到观测值，拼音。联系 HMM 模型，汉字“我”到汉字“爱”的过程是一个转移过程，如果用二元语法模型，则汉字“爱”在“我”的出现情况下有一个转移概率，如后汉字分析同理。同时，汉字我到拼音“wo”有一个发射过程，也有一个概率。则上图可变化为基于概率的识别流程图，如下。

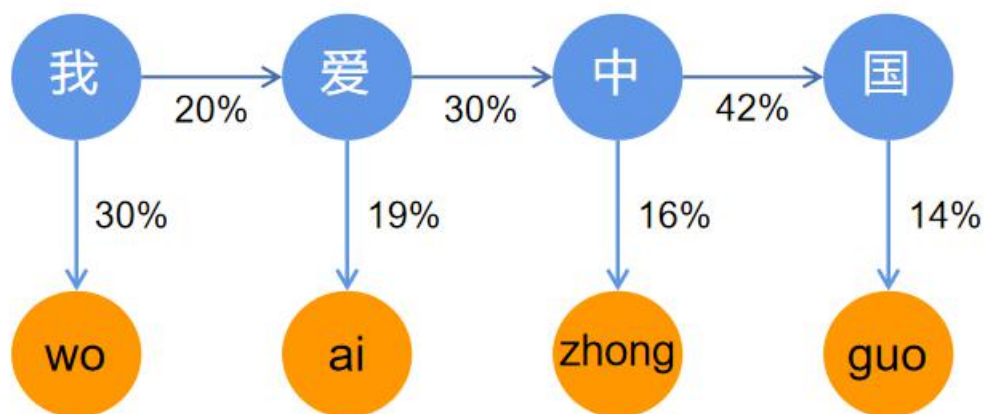


图2 我爱中国汉字拼音输入基于概率的识别过程

但是，拼音“wo”的对应汉字处理“我”还有“卧”、“窝”等，拼音“ai”对应汉字处理“爱”，还有“哎”，“唉”等，那么“wo ai”的组成情况就还会出现“卧爱”、“窝爱”、“我哎”等。如次就出现了如下图3的基于隐马尔科夫模型HMM的拼音转汉字模型图。

如次，基于HMM模型的解码问题，可以求解出状态转移链中概率最大的一条路径，此条路径即所求的汉子序列。

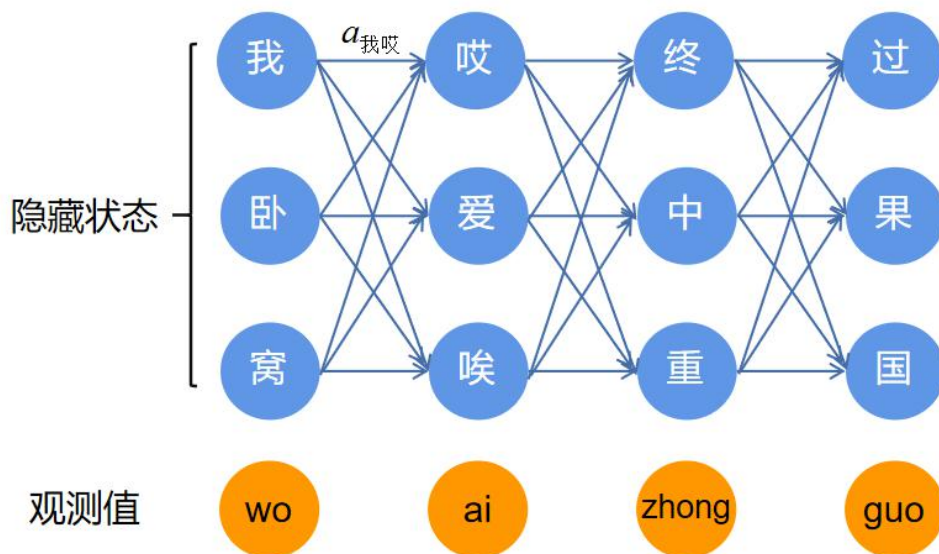


图3 “wo ai zhong guo” 拼音输入基于 HMM 的识别过程

### (三) 维特比算法

维特比(Viterbi)算法用于解码，在给定模型  $\mu$  和观察序列  $O$  的条件下，使条件概率  $P(Q|O, \mu)$  最大的状态序列，即

$$\hat{Q} = \operatorname{argmax}_Q P(Q | O, \mu)$$

维特比算法运用动态规划的搜索算法求解这种最优状态序列。为了实现这种搜索，首先定义一个维特比变量  $\delta_t(i)$ 。

维特比变量  $\delta_t(i)$  是在时间  $t$  时，HMM 沿着某一条路径到达状态  $s_i$ ，并输出观察序列的最大概率：

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1, q_2, \dots, q_t = s_i, O_1 O_2 \dots O_t | \mu)$$

$\delta_t(i)$  有如下递归关系：

$$\delta_{t-1}(i) = \max_j [\delta_t(j) \cdot a_{ji}] \cdot b_i(O_{t-1})$$

这种递归关系使我们能够运用动态规划搜索技术。为了记录在时间  $t$  时，HMM 通过哪一条概率最大的路径到达状态  $s_i$ 。

其伪代码如下：

---

**维特比算法(Viterbi algorithm)**

**初始化：**

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N$$

**归纳计算**

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t), \quad 2 \leq t \leq T; 1 \leq j \leq N$$

**终结**

$$\hat{Q}_T = \operatorname{argmax}_{1 \leq i \leq N} [\delta_T(i)]$$

$$\hat{P}(\hat{Q}_T) = \max_{1 \leq i \leq N} [\delta_T(i)]$$


---

#### (四) 模型实现与构建

其构建的流程如下图。

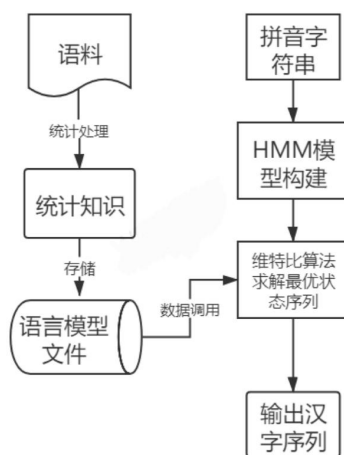


图 4 项目模型构件图

## 1. 语言模型训练

本处使用老师给的 toutiao\_cat\_data.txt 文件中的数据。

### (1) 语料清洗

由于文件中的文字段都是带有非法字符和大段文字的。所以这里本人使用了正则表达式来匹配中文字符，对于非中文字符都直接忽略，并切断句。

```
# toutiao_data
phrase = open('../train_data/toutiao_cat_data.txt', 'r', encoding="utf-8")
ans = []
for line in phrase.readlines():
    ls = line.split("_!")
    # 清洗数据
    ls = ls[3:]
    for item in ls:
        string = ""
        i = 0
        while(i < len(item)):
            res = re.match(r'[\u4E00-\u9FA5]', item[i])
            if(res == None):
                if(string != ""):
                    ans.append(string)
                    string = ""
            else:
                string += item[i]
            i += 1
        if(string != ""):
            ans.append(string)
```

如此，即可得到如图 5 所示的规则汉字串。

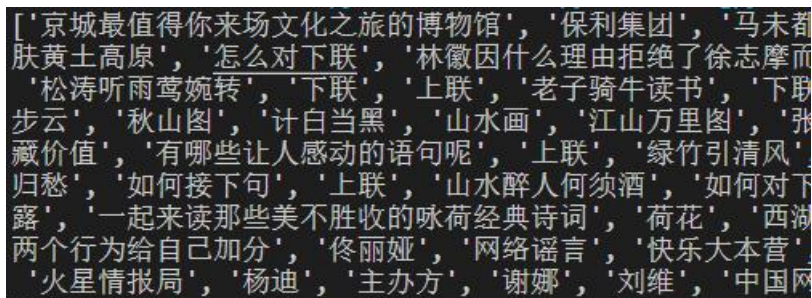


图 5 清洗非法字符后的汉字串

但是，HMM 模型需要拼音到汉字的发射概率，现在我们还缺少汉字的标准注音。所以，本人调用了 pypinyin 第三方库来对所有的汉字注音。获得拼音列表。

```

# 汉字的拼音添加
ans_pinyin = []
for item in ans:
    pinyin_ans = pinyin(u'{0}'.format(item), style=pypinyin.NORMAL)
    if(pinyin_ans == None):
        continue
    string = ""
    for item in pinyin_ans:
        string += item[0]+" "
    string = string[0:-1]
    if(string != ""):
        ans_pinyin.append(string)
# ['bao li ji tuan', 'ma wei du', 'zhong guo ke xue ji zhu guan',...]

```

## (2) 语言模型训练

基于如上的大量语料库文本，由一段文字可以得到文字库和拼音库，然后统计文字的频次、文字到拼音的频次、文字到文字的频次。如此，根据 N-Gram 语言模型原理

$$p(a_i | a_{i-1} \dots a_{i-n+1}) = \frac{c(a_i a_{i-1} \dots a_{i-n+1})}{\sum_{a_i} c(a_i a_{i-1} \dots a_{i-n+1})}$$

得出一元语言模型和二元语言模型：

$$p(a_i) = \frac{c(a_i)}{\sum_{a_i} c(a_i)}$$

$$p(a_i | a_{i-1}) = \frac{c(a_i a_{i-1})}{\sum_{a_i} c(a_i a_{i-1})}$$

同时训练时采用加一平滑技术得到如下公式：

$$p(a_i) = \frac{c(a_i) + 1}{|V| + \sum_{a_i} c(a_i)}$$

$$p(a_i | a_{i-1}) = \frac{c(a_i a_{i-1}) + 1}{|V| + \sum_{a_i} c(a_i a_{i-1})}$$

依照如上公式带入语料库文本信息则可以训练得到 HMM 模型的概率矩阵。同时，为方便 HMM 模型程序调用整理好的语料数据，将这些加工后的语料文件保存为 python 易使用的 npy 文件。

此外，为了方便使用如上的语言模型计算，这里保存了四个语料文件，分别为每一个拼音对应的所有已知汉字集合文件 py2hanzi.npy、为汉字编码后的汉字



编码字典文件 my\_hanzi\_dict.npy、汉字编码到汉字编码二字词映射频次对应的二维矩阵文件 my\_moving\_array.npy、单个汉字出现次数的列表文件 my\_hanzi\_num.npy 和汉字对应的各个拼音文件 my\_emission\_dic。

其生成的代码如下：

统计汉字的出现频次，用于计算一元语言模型。

```
# 统计汉字信息 [汉字] [出现次数]一元语料
hanzi_ls = []
hanzi_count_ls = []
for item in ans:
    for chr in item:
        if chr not in hanzi_ls:
            hanzi_ls.append(chr)
            hanzi_count_ls.append(0)
        else:
            hanzi_count_ls[hanzi_ls.index(chr)] += 1
np.save("../data/my_hanzi_num", hanzi_count_ls, allow_pickle=True,
fix_imports=True)
total_hanzi_num = len(hanzi_ls)
print(total_hanzi_num)
```

汉字的编码列表

```
# dic 汉字: 汉字编码 映射表
hanzi_dict = {}
encode_num = 0
for item in hanzi_ls:
    hanzi_dict[item] = encode_num
    encode_num += 1
np.save("../data/my_hanzi_dict", hanzi_dict, allow_pickle=True,
fix_imports=True)
```

汉字编码到汉字编码的次数映射，用于统计二元语法模型。

```
# 二元语料训练
# 汉字编码到汉字编码的映射 次数
hanzi_matrix = np.zeros([total_hanzi_num, total_hanzi_num])
for item in ans:
    for i in range(1, len(item)):
        chr1 = item[i-1]
        chr2 = item[i]
        code1 = hanzi_dict[chr1]
        code2 = hanzi_dict[chr2]
        hanzi_matrix[code1][code2] += 1
np.save("../data/my_moving_array", hanzi_matrix, allow_pickle=True,
fix_imports=True)
```

汉字到拼音的频数，其格式为{'了': {'le': 5, 'liao': 10}, '屈': {'qu': 5}}。

```

# 汉字拼音字典表数据准备
py2hanzi = {}
for i in range(len(ans)):
    pinyin_ls = ans_pinyin[i].split() # 拼音序列
    for pinyin_item in pinyin_ls: # 初始化
        py2hanzi[pinyin_item] = ""
for i in range(len(ans)):
    str = ans[i] # 汉字串
    pinyin_ls = ans_pinyin[i].split() # 拼音序列
    for j in range(len(str)):
        chr = str[j]
        pinyin_item = pinyin_ls[j]
        if( chr not in py2hanzi[pinyin_item]):
            py2hanzi[pinyin_item] += chr
np.save("../data/py2hanzi.npy", py2hanzi, allow_pickle=True,
fix_imports=True)
print(py2hanzi.keys())

# 汉字 对应的 拼音 频数 eg: {'了':{'le':5, 'liao':10}, '屈':{'qu':5}}
hanzi2pin_dict = {} # {'了':{'le':5, 'liao':10}, '屈':{'qu':5}}
hanzi_str = hanzi_ls # 已存在拼音的汉字序列
py_data_ls = [] # 已存在的拼音列表
# 构建双重字典表结构
for k,v in py2hanzi.items():
    py_data_ls.append(k)
    if "ü" in k:
        print(k)
    hanzi_str += v
    for chr in v:
        hanzi2pin_dict[chr] = {} # 初始化每一个字对应一个拼音频率字典
for k,v in py2hanzi.items():
    for chr in v:
        hanzi2pin_dict[chr][k] = 0 # 初始化每一个拼音的频率value
for i in range(len(ans)):
    str = ans[i] # 汉字串
    pinyin_ls = ans_pinyin[i].split() # 拼音序列
    for i in range(len(pinyin_ls)): # 去除音调
        pinyin_ls[i] = pinyin_ls[i][0:-1]
    if(len(str) != len(pinyin_ls)): # 拼音与汉字数不匹配
        continue
    for i in range(len(str)):
        if(str[i] not in hanzi_str): # 该汉字不在有拼音的汉字列表 添加到dic
            hanzi2pin_dict[ str[i] ] = {}
            hanzi2pin_dict[ str[i] ][ pinyin_ls[i] ] = 0
        py_data_ls = hanzi2pin_dict[ str[i] ].keys() # 该汉字所有的拼音列表
        if(pinyin_ls[i] not in py_data_ls):
            hanzi2pin_dict[ str[i] ][ pinyin_ls[i] ] = 0
            hanzi2pin_dict[str[i]][pinyin_ls[i]] += 1
phrase.close()
np.save("../data/my_emission_dic", hanzi2pin_dict, allow_pickle=True,
fix_imports=True)

```

至此，所有的语料文件都已训练好并保存为 npy 格式的文件，方便调用。



## 2. HMM 模型的构建

依照图 4 的模型，首先构建一个 HMM 的模型模块，输入为一串拼音序列，并判断“l”、“n”遇上元音“ü”的情况，替换“v”为“ü”，然后加载语料数据，构建 HMM 模型如图 3，每一个汉字状态圆圈代表一个节点，节点存储此汉字和汉字对应拼音的语料数据。然后通过维特比算法获得最优的汉字序列，输出汉字序列。其伪代码如下：

---

输入：拼音序列  $O_1O_2...O_T$

过程：

- 1: 加载语料数据文件，获得汉字到汉字、拼音的概率表和汉字的概率表
  - 2: 输入拼音
  - 3: 若“l”、“n”遇上元音“ü”的替换“v”为“ü”
  - 4: 初始化每个拼音的汉字状态节点，汉字状态节点初始化三类概率
  - 5: 维特比算法求解
  - 6: 输出汉字序列
- 

下面我们代码实现如上的伪代码。

首先是加载语料数据文件。

```
# init数据
# dic 从写好的numpy文件中读取字典如 'a': '啊啊啊钢吡腌嘎'
py2hanzi = np.load('../data/py2hanzi.npy', allow_pickle=True).item()
# dic 汉字: 汉字编码 映射表
hanzi_dict = np.load('../data/my_hanzi_dict.npy', allow_pickle=True).item()
# array 汉字到汉字的二维表，填充出现次数
hanzi_matrix = np.load('../data/my_moving_array.npy', allow_pickle=True) # 汉字出现次数
# array 汉字编码: 汉字出现次数
hanzi_num = np.load('../data/my_hanzi_num.npy', allow_pickle=True) # 汉字出现次数
total_num = sum(hanzi_num)
# 汉字 对应的 拼音 频数 eg: {'了': {'le': 5, 'liao': 10}, '屈': {'qu': 5}}
hanzi2pin_dict = np.load('../data/my_emission_dic.npy',
allow_pickle=True).item()
```

然后是设计了一个 Graph 邮箱图类来存储 HMM 模型的结构。其初始化为将输入的拼音字符串分解，为每个拼音构建其汉字节点，然后将每个节点初始化。

```

class Graph(object): # 有向图
    def __init__(self, pinyins):
        """
        @describe: 根据拼音所对应的所有汉字组合, 构造有向图
        @pinyins: 预测的拼音输入
        """
        self.bp_array = [] # 存储不同的拼音的汉字列表 [[],[],[...]
        self.pinyins = pinyins # 存储此HMM模型的可见拼音序列
        for py in pinyins:
            level = [] # 存储同一拼音下的多个汉字节点
            # 从拼音、汉字的映射表中读取汉字的出现次数以及汉字的词组数列
            hanzi_list = py2hanzi.get(py, "NONE")
            if(hanzi_list == "NONE"):
                print("[ERROR] Don't find the pinyin for:", py)
                continue
            for hanzi in hanzi_list:
                code = hanzi_dict[hanzi] # 汉字与编码映射表
                count = hanzi_num[code] # 汉字出现次数
                express = hanzi_matrix[code][:] # 该汉字编码到所有汉字编码的出现次数

                node = GraphNode(hanzi, count, express) # 生成有向图节点
                level.append(node)
            self.bp_array.append(level)

```

其中每个汉字都是一个节点，节点也是一个类，保存了其汉字、次数、维特比变量和前一个节点等熟悉。

```

class GraphNode(object):
    def __init__(self, hanzi, count, express):
        """
        @describe: 生成有向图节点
        @hanzi: 汉字与编码映射表
        @count: 汉字出现次数
        @express
        """
        # 当前节点所代表的汉字（即状态）
        self.hanzi = hanzi
        # 当前汉字的出现次数
        self.count = count
        # 当前汉字与其他汉字共同出现的次数（词组数列）
        self.express = express
        # 最优路径时，从起点到该节点的最高分， 维特比变量的值
        self.max_prob = 0.0
        # 最优路径时，该节点的前一个节点，用来输出路径的时候使用
        self.prev_node = None

```

在初始化好 HMM 模型结构后，我们就可以开始用维特比算法计算每一个节点的维特比变量，来对整个模型的节点进行遍历计算。其中初始节点的维特比变量用一元语法模型计算，后续节点用二元语法模型状态，都是用的加一平滑技术。然后发射概率也是加一平滑的。

```

def viterbi_i(i, graph):
    """
    @describe: 计算每一个可见值对应所有状态的维特比变量值
    @i: 表示第几个可见的值, graph二维数组的第一层index
    @graph: 二维节点数组,HMM模型的有向图,存储bp_array背包二维数组
    """
    # 初始化
    # 对于有向图,在第i层求所有节点的到该节点的最大概率
    if i == 0: # 如果为第0层
        for state_node in graph.bp_array[i]: # 遍历该拼音对应的所有状态(中文字符) 计算概率
            code_j = hanzi_dict[state_node.hanzi] # 第i个状态的j节点的编码
            num_j = hanzi_num[code_j] # 获得此汉字的一元语言模型的出现次数
            # 计算此模型节点的一元语言模型概率,加一平滑
            P_start = (1 + num_j) / (total_num + len(hanzi_num))
            # 计算此汉字到拼音i的发射概率
            # = find_P_emission(state_node.hanzi, graph.pinyins[i])
            P_emission = 1
            state_node.max_prob = P_start * P_emission
        return

    for state_node in graph.bp_array[i]:
        # 对于第j个节点,需要与前面第i-1层的所有节点匹配,求最大概率
        prob = []
        code_j = hanzi_dict[state_node.hanzi] # i层j节点的编码
        num_j = hanzi_num[code_j]

        # 此汉字到拼音i的发射概率
        P_emission = 1 # = find_P_emission(state_node.hanzi, graph.pinyins[i])

        for node_k in graph.bp_array[i-1]: # 对于第i-1层的k节点
            code_k = hanzi_dict[node_k.hanzi] # 上一层节点的编码
            num_k = hanzi_num[code_k] # 获得上一层节点汉字的频数(次数)
            a = (node_k.express[code_j] + 1) / \
                (num_k + len(hanzi_num)) # 转移概率 加一平滑
            P_moving = a * node_k.max_prob # 转移概率 * 前节点的维特比算子值
            prob.append(P_moving)

        # 获取最大概率在i-1层的位置
        max_k = prob.index(max(prob))
        state_node.max_prob = prob[max_k] * P_emission # 前驱最大的概率 * 发射概率
        state_node.prev_node = graph.bp_array[i-1][max_k] # 此状态节点的前驱状态节点
    return

def Viterbi(graph):
    symbol_num = len(graph.bp_array) # 有n个拼音符号
    for i in range(symbol_num): # 遍历每一个观测值——拼音
        viterbi_i(i, graph) # 计算每一个状态(中文字符)的实现概率

```

在训练完成后, HMM 模型就可以找到最优的路径了, 这时候利用每一个节点保存的最优上一节点可以反向遍历得到最佳的路径, 也就是最优的字符串序列, 期待吗如下。

```

def bestpath(graph): # 获取最佳路径
    symbol_num = len(graph.bp_array)
    max_prob = []
    for node in graph.bp_array[symbol_num-1]:
        max_prob.append(node.max_prob)
    max_index = max_prob.index(max(max_prob))
    node = graph.bp_array[symbol_num-1][max_index]
    result = []
    real_result = ''
    while True:
        result.append(node.hanzi)
        node = node.prev_node
        if node is None:
            break
        if node.prev_node is None:
            result.append(node.hanzi)
            break
    while len(result) > 0:
        hz = result.pop()
        real_result += hz
    return real_result

```

至此，HMM 模型已经训练完成和可以进行输入法预测功能。

## （五）模型改进

在如上语料模型训练中，存在一些问题，最终会大幅度影响整个 HMM 模型的训练结果。其缺点主要有以下几点：

1. 语料文件质量差。文中有大量非法字符，不得不用正则表达式去判断非法字符的位置，然后切断汉字串为两个汉字串。比如“我爱穿 T 恤，我好开心”，这里面有非法字符“T”和“，”，会将字符串切分为 3 个字符串段。

2. 语料文件无拼音注释，第三方库添加错误率较高。因为语料文件没有拼音语料，所以不得不用第三方库 Pypinyin 来为所有的汉字串注音，但是此 pypinyin 可以的拼音准确度会直接影响我们的概率，同时其中有非常多的识别错误，也没有声母 n、l 跟韵母 u 遇上的转换等过程。

3. 语料文件中的汉字数量只有 3000 多字，而常用的汉字有 8000 字左右，相差较多，同理，其拼音数量也严重缺失，导致语言模型训练会出现大量的平滑。

由此，本文在网上下载了一些第三方较好的语料库文件，并训练好用来做出更好的改进。同时，本文也在网上找了更丰富训练集来测试模型。在此，感谢 [https://github.com/THUzhangga/HMM\\_shurufa/tree/master/data](https://github.com/THUzhangga/HMM_shurufa/tree/master/data) 的语料文件



和测试集的帮助。

#### 四、实验结果及分析

##### 1. 基于实验语料文件的 HMM 训练与测试的结果

(1) 基于 toutiao\_cat\_data.txt 文件训练后，对于实验给的测试集.txt 文件进行测试如下图。可得到测试准确率在 79.2%。

```
result: 今天晚上有好看的电影 correct_ratio: 1.0
result: 北京奥运会开幕式非常精彩 correct_ratio: 1.0
result: 全国人民代表大会在北京人民大会隆重举行 correct_ratio: 0.95
result: 仅用的武侠小说非常精彩 correct_ratio: 0.81818181818182
result: 你的世界会变得更精彩 correct_ratio: 1.0
result: 深度学习技术推动了人工智能的发展 correct_ratio: 1.0
result: 在天安门广场举行乐隆中的阅兵式 correct_ratio: 0.866666666666667
result: 罗纪大数有些常用的集本公式 correct_ratio: 0.6923076923076923
result: 请大家选择你觉得可以的时间 correct_ratio: 1.0
result: 具有两好的钩统能力和交流能力 correct_ratio: 0.7857142857142857
result: 人在约都是世葱做到游住子都入的 correct_ratio: 0.3333333333333333
result: 见少可成的可是世十分科学的 correct_ratio: 0.5384615384615384
result: 我带着衣服黑筐眼睛 correct_ratio: 0.4444444444444444
result: 他钢琴弹的不错 correct_ratio: 0.8571428571428571
result: 小朋友们都喜欢去脚油 correct_ratio: 0.8
result: 这个晚举很有趣 correct_ratio: 0.7142857142857143
result: 他恨圣骑 correct_ratio: 0.25
result: 今天天气很好 correct_ratio: 1.0
result: 我们去三部霸 correct_ratio: 0.5
result: 大家可能不知道 correct_ratio: 1.0
result: 我们宿舍得等坏了 correct_ratio: 0.625
total correct ratio:79.20%
```

(2) 基于 toutiao\_cat\_data.txt 文件训练后，对于 github.com/THUzhangga 的 test\_set.txt 文件进行测试如下图。可得到测试准确率在 67.58%。

```
result: 我不知道款面的人为什么人的句子都那么奇怪 correct_ratio: 0.5
result: 这不是一条搜狗输入法都打不出来的句子 correct_ratio: 1.0
result: 业都收回天下 correct_ratio: 0.3333333333333333
result: 哈哈哈哈哈 correct_ratio: 0.8333333333333334
result: 林先生发疯了 correct_ratio: 1.0
result: 是不是一候可以用这个东西聊天 correct_ratio: 0.8571428571428571
total correct ratio:67.58%
```

##### 2. 基于改进后的语料文件的 HMM 训练与测试的结果

(1) 基于 github.com/THUzhangga 的语料文件训练后，对于实验给的测试集.txt 文件进行测试如下图。可得到测试准确率在 87.61%，可明显看到准确率大幅度提升。

```
result: 这个玩具很有趣 correct_ratio: 1.0
result: 他很生气 correct_ratio: 1.0
result: 今天天气很好 correct_ratio: 1.0
result: 我们去散步把 correct_ratio: 0.8333333333333334
result: 大家可能不知道 correct_ratio: 1.0
result: 我们宿舍的灯坏了 correct_ratio: 0.875
total correct ratio:87.61%
```

(2) 基于 github.com/THUzhangga 的语料文件训练后对于



github.com/THUzhangga 的 test\_set.txt 文件进行测试如下图。可得到测试准确率在 75.56%，可明显看到准确率大幅度提升。

```
result: 这不是一条搜狗输入法都大部出来的车子 correct_ratio: 0.8
result: 也都收回天下 correct_ratio: 0.3333333333333333
result: 哈哈哈哈哈 correct_ratio: 0.8333333333333334
result: 林先生发疯了 correct_ratio: 1.0
result: 是不是以后可以用这个东西聊天 correct_ratio: 1.0
total correct ratio:75.56%
```

3.模型评价与分析

本文选取一些测试样例和搜狗输入法做比较，可以明显观察到长句的错误率较高，短句和常见的词组正确率一般都较高，同时搜狗输入法的长句错误率也较高。

此外，在运行时间和资源消耗的上，时间预测一个拼音字符串的主要时间消耗在数据文件的加载中，同时存储开销达到百兆的大小，并随着语料库的增加还会继续增大。

总体来说，本项目的正确率较高，但是项目的其他资源消耗太大，仅作为实验验证还是不错的。

表 1 实验项目结果对比

预测结果	搜狗输入法	正确答案
虽然已经解决了建立新积分方法的首要问题	虽然已经解决乐见离心机芬芳发的首要问题	虽然已经解决了建立新积分方法的首要问题
建立了交易办机上的策独立伦	建立了交易班级上的测度理论	建立了较一般集上的测度理论
后面我们将成具有这种性质的函数为可测寒暑	后面我们将成具有这种性质的函数味可测函数	后面我们将称具有这种性质的函数为可测函数
今天也是好天气	今天也是好天气	今天也是好天气
中央已经决定了	中央已经决定了	中央已经决定了