哈尔滨工业大学 2020 年秋季学期

计算学部本科生
“中文信息处理”课
大作业

作 业 题 目：中文命名实体识别

姓　　名：梅智敏

学　　号：1183710118

学 生 专 业：软件工程

任 课 教 师：刘秉权

2020 年 10 月 26 日

中文信息处理实验二

一、实验内容

中文名实体识别

二、实验要求

- 使用任意方法实现任一类中文名实体识别；
- 给定足够规模的测试文本，在其上标注至少100个实体识别结果（以附件形式提供）；
- 计算出实体识别的准确率和召回率，并给出计算依据；
- 针对识别结果中存在的问题给出具体分析；
- 提交实验报告，给出详细实验过程、结果和结论；提交源代码、可执行程序和程序中使用的其他资源。

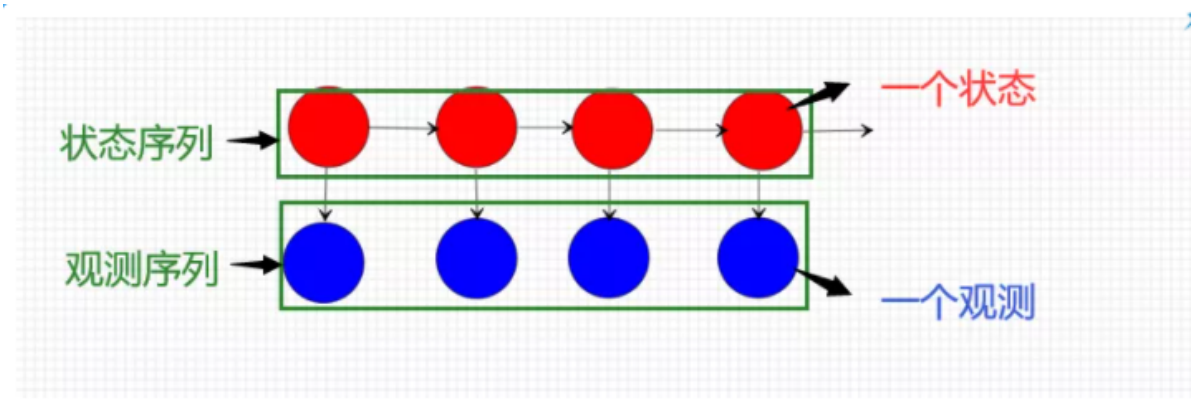
三、实验环境

win10、python3.7.4、pycharm2020.2

四、数学原理

因为实验不限方法，本实验使用统计机器学习方法中的**HMM**模型进行求解，故下面对HMM进行介绍。

> **隐马尔可夫模型是关于时序的概率模型，描述由一个隐藏的马尔可夫链随机生成不可观测的状态随机序列，再由各个状态生成一个观测从而产生观测随机序列的过程。隐藏的马尔可夫链随机生成的状态的序列，称为状态序列；每个状态生成一个观测，而由此产生的观测的随机序列，称为观测序列。序列的每一个位置又可以看作是一个时刻。**



隐马尔可夫模型**HMM**由**初始概率分布**π、**状态转移概率分布**A以及**观测概率分布**B确定，它们称为三要素。

HMM的形式定义如下

$$Q=\{q_1,q_2\dots q_N\},\ V=\{v_1,v_2\dots v_M\}$$

Q表示所有可能的状态集合，V表示所有可能的观测集合；N是所有可能的状态数，M是所有可能的观察数。

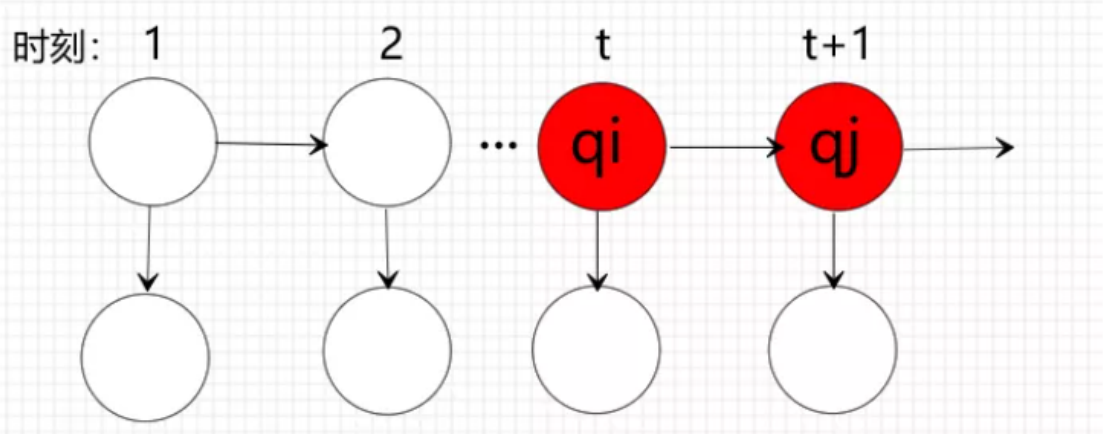
现在有长度为T的状态序列和观测序列

$$I=\{i_1,i_2\dots i_T\},\ O=\{o_1,o_2\dots o_T\}$$

A是状态转移概率矩阵

$$A = [a_{ij}]_{N \times N}$$

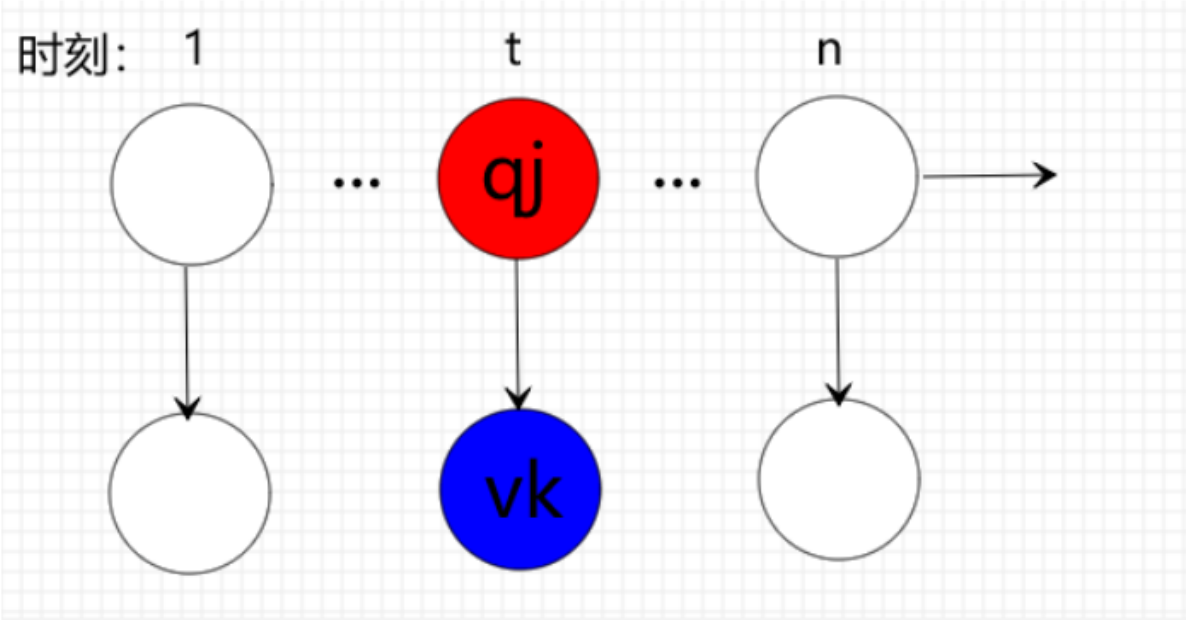
其中a_{ij}表示从当前状态q_i转移到下一状态q_j的概率



B是观测概率矩阵

$$B = [b_{jk}]_{N \times M}$$

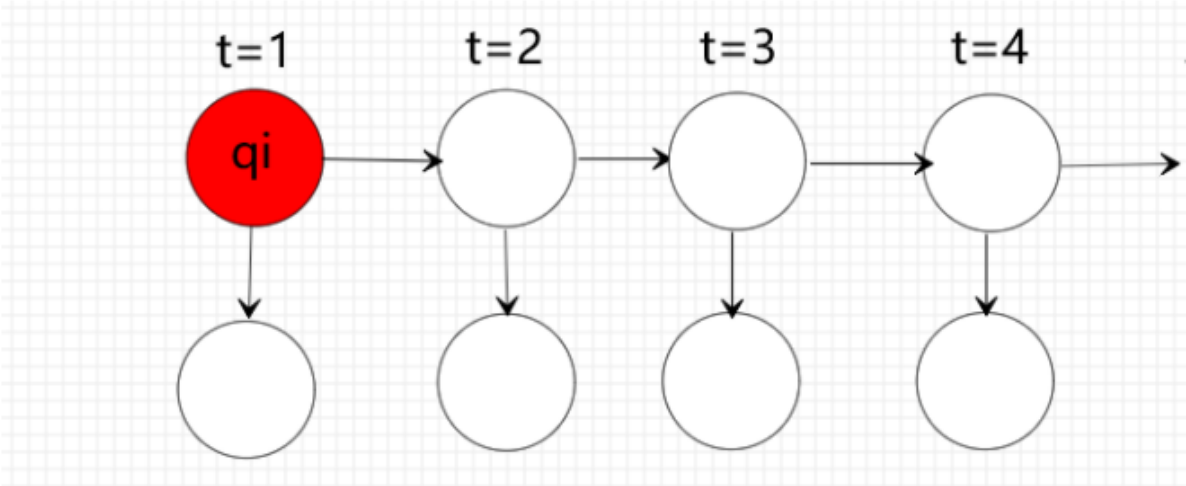
其中b_{jk}表示处于状态q_j的条件下产生观测o_k的概率



π是初始状态概率向量

$$\pi = (\pi_i)$$

其中π_i表示初始状态为q_i的概率



隐马尔可夫模型 λ 可以用三元符号表示

$$\lambda = (A, B, \pi)$$

HMM的基本假设

我们从HMM的形式定义就能很明显的看出，它满足下面的基本假设。

- 齐次马尔可夫性假设

隐藏的马尔可夫链在任意时刻t的状态只依赖于其前一时刻的状态，与其他时刻的状态及观测无关，也与时刻t无关

- 观测独立性假设

假设任意时刻的观测只依赖于该时刻的马尔可夫链的状态，与其他观测及状态无关

HMM用途

- 学习参数A, B, π

已知观测序列O，估计模型参数$\lambda = (A, B, \pi)$参数，使得$P(O|\lambda)$最大

- 预测状态序列

已知$\lambda = (A, B, \pi)$和观测序列O，求出隐藏的最有可能的状态序列I

五、主要算法介绍

根据前面数学原理部分的叙述，我们可以知道利用HMM进行中文命名实体识别的核心步骤为

- 从训练集中学习出A, B, π
- 利用学习到的参数对测试集进行预测

本实验的数据集采用的是BIOES标注法

5.1 从训练集上学习出参数

首先定义出一些基本量，并初始化三个参数

```
    # 表示所有可能的标签个数N
    self.num_tag = len(self.tag2id)
    # 所有字符（包括汉字）的Unicode编码个数
    self.num_char = 65535
    # 状态转移矩阵,N*N
    self.A = np.zeros((self.num_tag, self.num_tag))
    # 观测概率矩阵,N*M
    self.B = np.zeros((self.num_tag, self.num_char))
    # 初始隐状态概率,N
    self.pi = np.zeros(self.num_tag)
```

对于B的确定，核心思想是：训练集中每出现一个“序列——观测值”对，就将B的对应位置加1，待遍历完震整个训练集之后，再将B中的数据进行归一化处理，这样矩阵中每个数字即可用来表示"概率"。

核心代码如下：

```
        for i in tqdm(range(len(lines))):
            if len(lines[i]) == 1:
                # 空行，即只有一个换行符，跳过
                continue
            else:
                # split()的时候，多个空格当成一个空格
                cut_char, cut_tag = lines[i].split()
                # ord是python内置函数
                # ord(c)返回字符c对应的十进制整数
                self.B[self.tag2id[cut_tag]][ord(cut_char)] += 1
```

A和π的估计与B原理相同，不再赘述。

5.2 使用参数在测试集上进行预测

在已有A, B, π的前提下，对于给定的观测字符序列String，我们可以预测出它背后隐藏的标注序列，也就是隐式的HMM链。

本实验使用$viterbi$算法进行求解

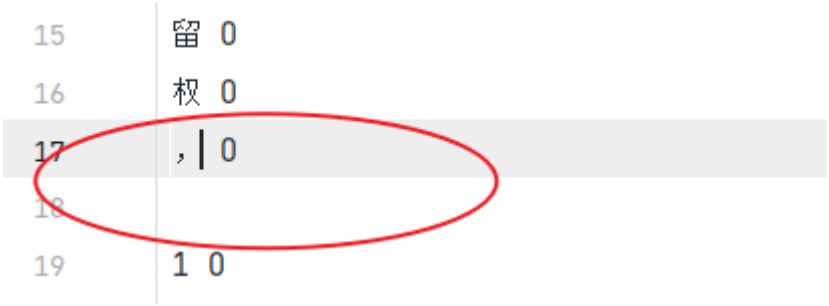
> 维特比算法是一种**动态规划算法**用于寻找最有可能产生观测事件序列的-维特比路径-隐含状态序列，特别是在马尔可夫信息源上下文和**HMM**中。术语“维特比路径”和“维特比算法”也被用于寻找观察结果最有可能解释相关的动态规划算法。
>
> 核心算法如下：

```
        for i in range(1, T):
            # arr.reshape(4,-1) 将arr变成4行的格式，列数自动计算的(c=4, d=16/4=4)
            temp = delta[i - 1].reshape(self.num_tag, -1) + self.A
            # 按列取最大值
            delta[i] = np.max(temp, axis=0)
            # 得到delta值
            delta[i] = delta[i, :] + self.B[:, ord(Obs[i])]
            # 取出元素最大值对应的索引
            psi[i] = np.argmax(temp, axis=0)
        # 最优路径回溯
        path = np.zeros(T)
        path[T - 1] = np.argmax(delta[T - 1])
        for i in range(T - 2, -1, -1):
            path[i] = int(psi[i + 1][int(path[i + 1])])
```

六、实验流程

6.1 数据集预处理

本实验的数据集来自互联网，标注方式为**BIOES**，但是原生数据集中有很多无用的换行符号，例如：



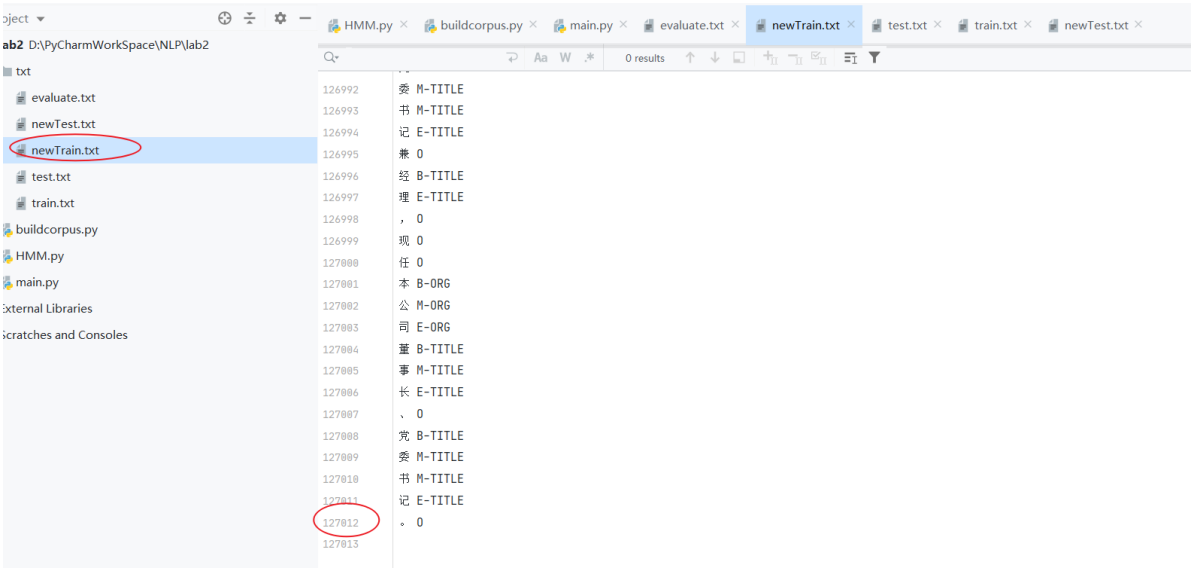
这里并不是一个句子的结尾，却有个无用的换行符(因为这个换行符并不是2个句子的分界线)。

因此我们需要先对源数据集进行预处理

```
            if wordList[-1] == '\n' and (", " in wordList[-2]):
                # 去掉多余的换行符号
                wordList.pop()
```

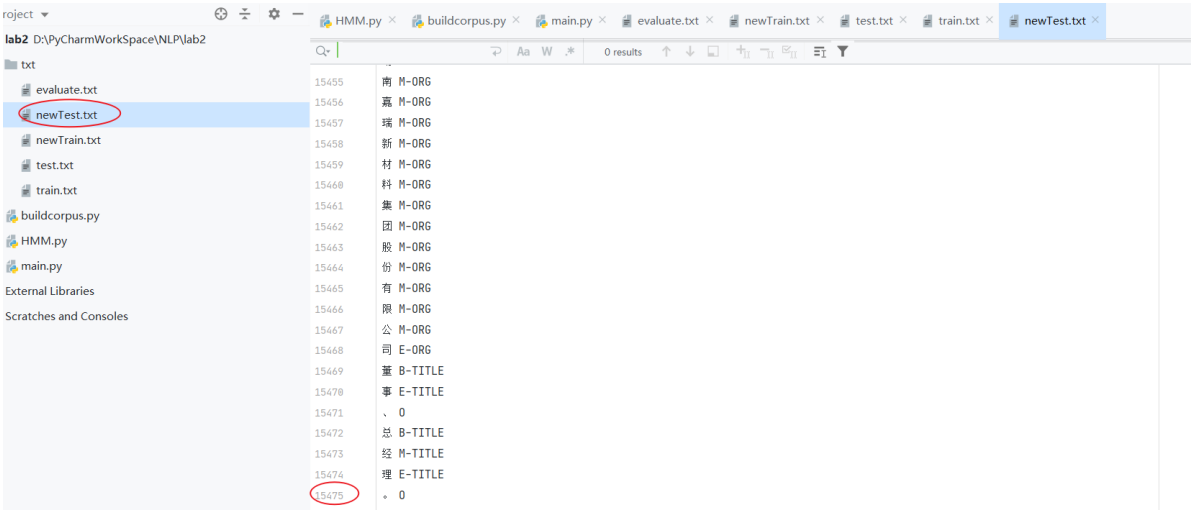
6.2 学习出参数

使用的数据集大小如下：



6.3 在测试集上预测并评估

测试集大小如下：



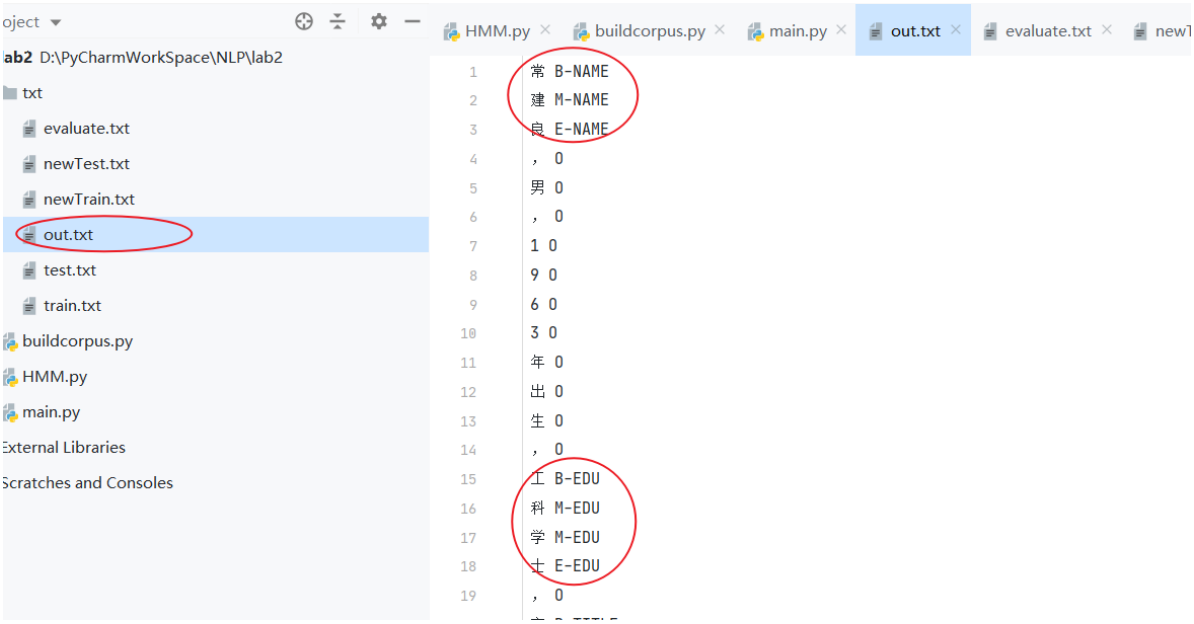
测试集后面的标注为$TrueAnswer$, 我们需要依据此来计算准确率和召回率。

在进行预测之前，我们需要先对测试集进行读取，获得所有的句子列表$wordList$和对应的标注列表$tagList$

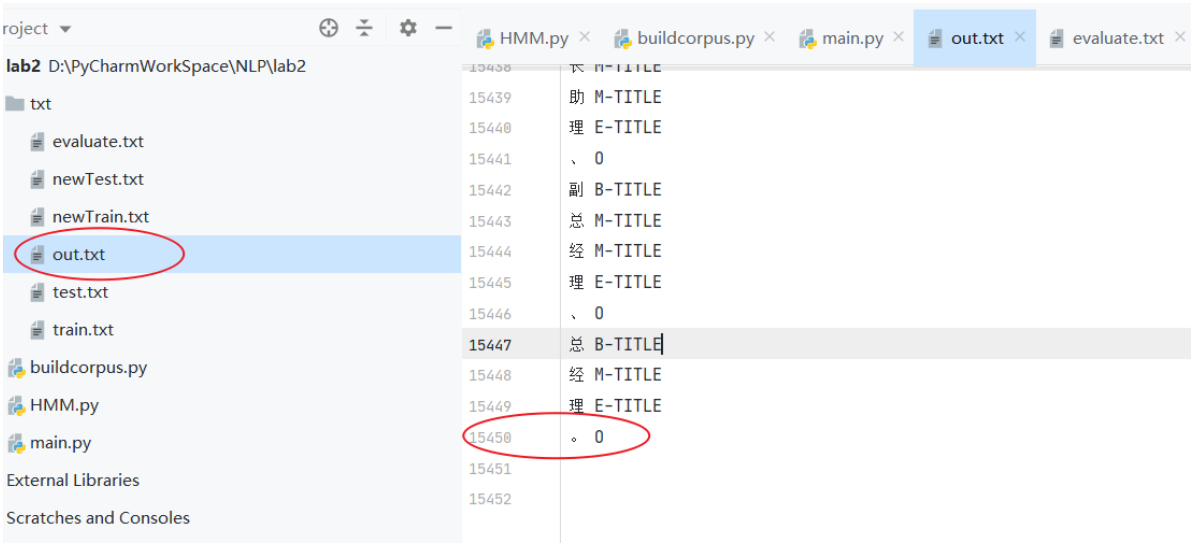
```
def build_corpus(inPath):
    """
    读取txt文件，获取wordList和tagList
    :param inPath:
    :return:
    """
    word_lists = []
    tag_lists = []
    with open(inPath, 'r', encoding='utf-8') as f:
        word_list = []
        tag_list = []
        for line in f:
            if line != '\n':
                word, tag = line.strip('\n').split()
                word_list.append(word)
                tag_list.append(tag)
            else:
                word_lists.append(word_list)
                tag_lists.append(tag_list)
                word_list = []
                tag_list = []

        return word_lists, tag_lists
```

再分别对每个句子进行预测，从而得到预测的标注列表$predictedTagList$，将结果输出到txt文件



可见结果还是不错的，$out.txt$文件共1w5k行



下面计算准确率和召回率

以$tag = E - EDU$为例

准确率：

$$p = \frac{Number(TrueTagList[i][j] == E - EDU)}{Number(PredictedTagList[i][j] == E - EDU)}$$

召回率：

$$r = \frac{Number(PredictedTagList[i][j] == E - EDU)}{Number(TrueTagList[i][j] == E - EDU)}$$

尤其注意：**在使用上面的公式时，需要先找到分母中满足条件的(i,j)对，再将同样的(i,j)对代入分子。并不是遍历全部的(i,j)对空间。**

输出到txt文件：

```
E-EDU: 准确率： 0.9016393450685299召回率： 0.9821428573022959
B-RACE: 准确率： 0.9285714336734691召回率： 0.9285714336734691
E-TITLE: 准确率： 0.9500640205504942召回率： 0.9636363636835892
B-NAME: 准确率： 0.9811320756496974召回率： 0.9285714292091837
M-NAME: 准确率： 0.8860759508091651召回率： 0.8536585383700178
M-CONT: 准确率： 0.9464285723852041召回率： 1.0
M-ORG: 准确率： 0.9031095088095742召回率： 0.929499072372565
B-CONT: 准确率： 0.9333333355555555召回率： 1.0
B-EDU: 准确率： 0.8852459025799516召回率： 0.9642857146045918
B-LOC: 准确率： 0.28571438775508745召回率： 0.3333334444444259
B-ORG: 准确率： 0.8433734942454845召回率： 0.887681159623766
B-TITLE: 准确率： 0.8783610756999218召回率： 0.8909090910507674
E-CONT: 准确率： 0.9333333355555555召回率： 1.0
E-ORG: 准确率： 0.825862069265755召回率： 0.8677536234279826
E-NAME: 准确率： 0.8584905673727304召回率： 0.8125000016741071
M-TITLE: 准确率： 0.901340482626627召回率： 0.8750650703409344
E-LOC: 准确率： 0.5714286326530524召回率： 0.666666722222213
B-PRO: 准确率： 0.5217391408317579召回率： 0.72727273553719
M-LOC: 准确率： 0.615384644970412召回率： 0.3809524104308376
O: 准确率： 0.9590064620438193召回率： 0.9157346702823497
M-PRO: 准确率： 0.4230769286242603召回率： 0.6470588287197231
M-EDU: 准确率： 0.92473118320037召回率： 0.9608938549670735
E-PRO: 准确率： 0.6086956606805292召回率： 0.8484848530762167
E-RACE: 准确率： 0.9285714336734691召回率： 0.9285714336734691
S-RACE: 准确率： 3.3333322222225924e-07召回率： 1.0
S-NAME: 准确率： 1.428571224489825e-07召回率： 1.0
M-RACE: 准确率： 1.0召回率： 1.0
S-ORG: 准确率： 1.0召回率： 1.0
```

可见准确率和召回率还是很不错的

6.4 实验存在的问题

使用HMM的前提就是满足：

- 齐次马尔可夫性假设

隐藏的马尔可夫链在任意时刻t的状态只依赖于其前一时刻的状态，与其他时刻的状态及观测无关，也与时刻t无关

- 观测独立性假设

假设任意时刻的观测只依赖于该时刻的马尔可夫链的状态，与其他观测及状态无关

但是对于中文命名实体，并不严格满足这2个假设。

七、附录

7.1 main.py

```
import HMM
import buildcorpus as bc


def UpdateFile(inPath, outPath):
    wordList = []
    with open(inPath, 'r', encoding='utf-8') as f:
        lines = f.readlines()
        N = len(lines)
        wordList.append(lines[0])
        for i in range(N-1):
            word = lines[i+1]
            wordList.append(word)
            if wordList[-1] == '\n' and (", " in wordList[-2]):
                # 去掉多余的换行符号
                wordList.pop()
    with open(outPath, 'w', encoding='utf-8') as f2:
        for i in range(len(wordList)):
            f2.write(wordList[i])


def out(wordList, predictedTagList, outPath):
    N = len(wordList)
    for i in range(N):
        for j in range(len(wordList[i])):
            # 处理wordlist
            wordList[i][j] += " "+predictedTagList[i][j]

    with open(outPath, 'w', encoding='utf-8') as f:
        for m in range(N):
            for n in range(len(wordList[m])):
                f.write(wordList[m][n]+"\n")
            f.write("\n")


UpdateFile("./txt/train.txt","./txt/newTrain.txt")
UpdateFile("./txt/test.txt","./txt/newTest.txt")
hmm = HMM.HMM()
hmm.train("./txt/newTrain.txt")
TrueWordList, TrueTagList = bc.build_corpus("./txt/newTest.txt")

predictTagList = []
# 对每个句子进行解码，求出该句子预测的tag
for x in TrueWordList:
    tag = hmm.viterbi(x)
    # 加入预测的tagList中
    predictTagList.append(tag)
out(TrueWordList, predictTagList, "./txt/out.txt")
hmm.calculate(TrueTagList, predictTagList,"./txt/evaluate.txt")
```

7.2 HMM.py

```
import numpy as np
# 第三方进度条库
from tqdm import tqdm


class HMM:
    def __init__(self):
        # 标记-id
        self.tag2id = {'E-EDU': 0,
                       'B-RACE': 1,
                       'E-TITLE': 2,
                       'B-NAME': 3,
                       'M-NAME': 4,
                       'M-CONT': 5,
                       'M-ORG': 6,
                       'B-CONT': 7,
                       'B-EDU': 8,
                       'B-LOC': 9,
                       'B-ORG': 10,
                       'B-TITLE': 11,
                       'E-CONT': 12,
                       'E-ORG': 13,
                       'E-NAME': 14,
                       'M-TITLE': 15,
                       'E-LOC': 16,
                       'B-PRO': 17,
                       'M-LOC': 18,
                       'O': 19,
                       'M-PRO': 20,
                       'M-EDU': 21,
                       'E-PRO': 22,
                       'E-RACE': 23,
                       'S-RACE': 24,
                       'S-NAME': 25,
                       'M-RACE': 26,
                       'S-ORG': 27
                       }
        # id-标记
        self.id2tag = dict(zip(self.tag2id.values(), self.tag2id.keys()))
        # 表示所有可能的标签个数N
        self.num_tag = len(self.tag2id)
        # 所有字符的Unicode编码个数 x16
        self.num_char = 65535
        # 转移概率矩阵,N*N
        self.A = np.zeros((self.num_tag, self.num_tag))
        # 发射概率矩阵,N*M
        self.B = np.zeros((self.num_tag, self.num_char))
        # 初始隐状态概率,N
        self.pi = np.zeros(self.num_tag)
        # 无穷小量
        self.epsilon = 1e-100

    def train(self, corpus_path):
        '''
```

```
        函数功能：通过数据训练得到A、B、pi
        :param corpus_path: 数据集文件路径
        :return: 无返回值
        '''
        with open(corpus_path, mode='r', encoding='utf-8') as f:
            # 读取训练数据
            lines = f.readlines()
        print('开始训练数据：')
        for i in tqdm(range(len(lines))):
            if len(lines[i]) == 1:
                # 空行，即只有一个换行符，跳过
                continue
            else:
                # split()的时候，多个空格当成一个空格
                cut_char, cut_tag = lines[i].split()
                # ord是python内置函数
                # ord(c)返回字符c对应的十进制整数
                self.B[self.tag2id[cut_tag]][ord(cut_char)] += 1
                if len(lines[i - 1]) == 1:
                    # 如果上一个数据是空格
                    # 即当前为一句话的开头
                    # 即初始状态
                    self.pi[self.tag2id[cut_tag]] += 1
                    continue
                pre_char, pre_tag = lines[i - 1].split()
                self.A[self.tag2id[pre_tag]][self.tag2id[cut_tag]] += 1
        # 为矩阵中所有是0的元素赋值为epsilon
        self.pi[self.pi == 0] = self.epsilon
        # 防止数据下溢,对数据进行对数归一化
        self.pi = np.log(self.pi) - np.log(np.sum(self.pi))
        self.A[self.A == 0] = self.epsilon
        # axis=1将每一行的元素相加，keepdims=True保持其二维性
        self.A = np.log(self.A) - np.log(np.sum(self.A, axis=1, keepdims=True))
        self.B[self.B == 0] = self.epsilon
        self.B = np.log(self.B) - np.log(np.sum(self.B, axis=1, keepdims=True))
        print('训练完毕！')

    def viterbi(self, Obs):
        """
        函数功能：使用viterbi算法进行解码
        :param Obs: 要解码的中文String
        :return: 预测的tagList
        """
        # 获得观测序列的文本长度
        T = len(Obs)
        # T*N
        delta = np.zeros((T, self.num_tag))
        # T*N
        psi = np.zeros((T, self.num_tag))
        # ord是python内置函数
        # ord(c)返回字符c对应的十进制整数
        # 初始化
        delta[0] = self.pi[:] + self.B[:, ord(Obs[0])]
        # range（）左闭右开
        for i in range(1, T):
            # arr.reshape(4,-1) 将arr变成4行的格式，列数自动计算的(c=4, d=16/4=4)
            temp = delta[i - 1].reshape(self.num_tag, -1) + self.A
            # 按列取最大值
```

```
            delta[i] = np.max(temp, axis=0)
            # 得到delta值
            delta[i] = delta[i, :] + self.B[:, ord(Obs[i])]
            # 取出元素最大值对应的索引
            psi[i] = np.argmax(temp, axis=0)
        # 最优路径回溯
        path = np.zeros(T)
        path[T - 1] = np.argmax(delta[T - 1])
        for i in range(T - 2, -1, -1):
            path[i] = int(psi[i + 1][int(path[i + 1])])

        tagList = []
        for i in range(len(path)):
            tagList.append(self.id2tag[path[i]])
        return tagList

    def calculate(self, TrueTagList, PredictedTagList, outFile):
        answer = []
        # 分别计算每种tag的准确率和召回率
        for tag in self.tag2id.keys():
            # 计算准确率
            denominator = 1e-6
            Numerator = 1e-6
            for i in range(len(PredictedTagList)):
                for j in range(len(PredictedTagList[i])):
                    if PredictedTagList[i][j] == tag:
                        denominator += 1
                        if TrueTagList[i][j] == tag:
                            Numerator += 1
            p = Numerator/denominator
            # 计算召回率
            denominator2 = 1e-6
            Numerator2 = 1e-6
            for i in range(len(TrueTagList)):
                for j in range(len(TrueTagList[i])):
                    if TrueTagList[i][j] == tag:
                        denominator2 += 1
                        if PredictedTagList[i][j] == tag:
                            Numerator2 += 1
            r = Numerator2/denominator2
            # 构建输出字符串
            string = tag+": "+"准确率：  "+str(p)+"召回率：  "+str(r)
            answer.append(string)

        with open(outFile, 'w', encoding='utf-8') as f:
            for i in range(len(answer)):
                f.write(answer[i]+"\n")
```

7.3 buildcorpus.py

```
def build_corpus(inPath):
    """
    读取txt文件，获取wordList和tagList
    :param inPath:
    :return:
```

```
    """
    word_lists = []
    tag_lists = []
    with open(inPath, 'r', encoding='utf-8') as f:
        word_list = []
        tag_list = []
        for line in f:
            if line != '\n':
                word, tag = line.strip('\n').split()
                word_list.append(word)
                tag_list.append(tag)
            else:
                word_lists.append(word_list)
                tag_lists.append(tag_list)
                word_list = []
                tag_list = []

        return word_lists, tag_lists
```