PJ2 Part1实验报告

HMM原理

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(state sequence)**; 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(observation sequence)**。序列的每一个位置又可以看作是一个时刻。

隐马尔可夫模型由**初始概率分布**、**状态转移概率分布**以及**观测概率分布**确定。隐马尔可夫模型的形式定义如下:
设Q 是所有可能的状态的集合，V 是所有可能的观测的集合:

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

其中，N 是可能的状态数，M 是可能的观测数
I 是长度为T的状态序列，O 是对应的观测序列:

$$I = (i_1, i_2, \cdots, i_T), \quad O = (o_1, o_2, \cdots, o_T)$$

A是状态转移概率矩阵：

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

其中，

$$a_{ij} = P(i_{t+1} = q_j | i_t = q_i), \quad i = 1, 2, \cdots, N; \quad j = 1, 2, \cdots, N$$

是在时刻 t 处于状态 q_i 的条件下在时刻 $t+1$ 转移到状态 q_j 的概率。

B 是观测概率矩阵：

$$B = [b_j(k)]_{N \times M}$$

其中，

$$b_j(k) = P(o_t = v_k | i_t = q_j), \quad k = 1, 2, \cdots, M; \quad j = 1, 2, \cdots, N$$

是在时刻 t 处于状态 q_j 的条件下生成观测 v_k 的概率。

π 是初始状态概率向量：

$$\pi = (\pi_i)$$

其中，

$$\pi_i = P(i_1 = q_i), \quad i = 1, 2, \cdots, N$$

是时刻 $t=1$ 处于状态 q_i 的概率。

我们的词性标注任务是监督学习，因为我们的训练集语句是标注过的，我们需要通过训练集合中的数据学习HMM模型的三个参数：**初始概率分布**、**状态转移概率分布**以及**观测概率分布**。由于是监督学习，所以这种学习实际上就是对训练集的统计过程。以中文为例，统计每个句子开头都是什么状态，可以得到初始概率分布；统计每个句子中相邻字符之间的状态转移，可以得到状态转移概率分布；统计每个句子每个状态对应什么字符，可以得到观测概率分布。

1. 转移概率 a_{ij} 的估计

设样本中时刻 t 处于状态 i 时刻 $t+1$ 转移到状态 j 的频数为 A_{ij}，那么状态转移概率 a_{ij} 的估计是

$$\hat{a}_{ij} = \frac{A_{ij}}{\sum_{j=1}^{N} A_{ij}}, \quad i = 1, 2, \cdots, N; \quad j = 1, 2, \cdots, N$$

2. 观测概率 $b_j(k)$ 的估计

设样本中状态为 j 并观测为 k 的频数是 B_{jk}，那么状态为 j 观测为 k 的概率 $b_j(k)$ 的估计是

$$\hat{b}_j(k) = \frac{B_{jk}}{\sum_{k=1}^{M} B_{jk}}, \quad j = 1, 2, \cdots, N; \quad k = 1, 2, \cdots, M \tag{10.31}$$

3. 初始状态概率 π_i 的估计 $\hat{\pi}_i$ 为 S 个样本中初始状态为 q_i 的频率

学习到HMM模型的三个参数以后，预测阶段要使用维特比算法进行解码，维特比算法实际是用动态规划(dynamic programming)解隐马尔可夫模型预测问题，即用动态规划求概率最大路径(最优路径)。这时一条路径对应着一个状态序列。

根据动态规划原理，最优路径具有这样的特性：如果最优路径在时刻 t 通过结点 i_t^*，那么这一路径从结点 i_t^* 到终点 i_T^* 的部分路径，对于从 i_t^* 到 i_T^* 的所有可能的部分路径来说，必须是最优的。因为假如不是这样，那么从 i_t^* 到 i_T^* 就有另一条更好的部分路径存在，如果把它和从 i_1^* 到达 i_t^* 的部分路径连接起来，就会形成一条比原来的路径更优的路径，这是矛盾的。依据这一原理，我们只需从时刻 $t=1$ 开始，递推地计算在时刻 t 状态为 i 的各条部分路径的最大概率，直至得到时刻 $t=T$ 状态为 i 的各条路径的最大概率。时刻 $t=T$ 的最大概率即为最优路径的概率 P^*，最优路径的终结点 i_T^* 也同时得到。之后，为了找出最优路径的各个结点，从终结点 i_T^* 开始，由后向前逐步求得结点 $i_{T-1}^*, \cdots, i_1^*$，得到最优路径 $I^*=(i_1^*, i_2^*, \cdots, i_T^*)$。这就是维特比算法。

首先导入两个变量 δ 和 Ψ。定义在时刻 t 状态为 i 的所有单个路径 $(i_1, i_2, \cdots, i_t)$ 中概率最大值为

$$\delta_t(i)=\max_{i_1, i_2, \cdots, i_{t-1}} P(i_t=i, i_{t-1}, \cdots, i_1, o_t, \cdots, o_1 | \lambda), \quad i=1,2,\cdots,N \tag{10.44}$$

由定义可得变量 δ 的递推公式：

$$\begin{aligned}\delta_{t+1}(i) &= \max_{i_1, i_2, \cdots, i_t} P(i_{t+1}=i, i_t, \cdots, i_1, o_{t+1}, \cdots, o_1 | \lambda) \\ &= \max_{1 \leqslant j \leqslant N}[\delta_t(j) a_{ji}] b_i(o_{t+1}), \quad i=1,2,\cdots,N; \ t=1,2,\cdots,T-1\end{aligned} \tag{10.45}$$

定义在时刻 t 状态为 i 的所有单个路径 $(i_1, i_2, \cdots, i_{t-1}, i)$ 中概率最大的路径的第 $t-1$ 个结点为

$$\Psi_t(i)=\arg \max_{1 \leqslant j \leqslant N}[\delta_{t-1}(j) a_{ji}], \quad i=1,2,\cdots,N \tag{10.46}$$

维特比算法步骤如下:

输入：模型 $\lambda=(A,B,\pi)$ 和观测 $O=(o_1,o_2,\cdots,o_T)$；

输出：最优路径 $I^*=(i_1^*,i_2^*,\cdots,i_T^*)$。

（1）初始化

$$\delta_1(i)=\pi_i b_i(o_1),\quad i=1,2,\cdots,N$$

$$\Psi_1(i)=0,\quad i=1,2,\cdots,N$$

（2）递推。对 $t=2,3,\cdots,T$

$$\delta_t(i)=\max_{1\leqslant j\leqslant N}[\delta_{t-1}(j)a_{ji}]b_i(o_t),\quad i=1,2,\cdots,N$$

$$\Psi_t(i)=\arg\max_{1\leqslant j\leqslant N}[\delta_{t-1}(j)a_{ji}],\quad i=1,2,\cdots,N$$

（3）终止

$$P^*=\max_{1\leqslant i\leqslant N}\delta_T(i)$$

$$i_T^*=\arg\max_{1\leqslant i\leqslant N}[\delta_T(i)]$$

（4）最优路径回溯。对 $t=T-1,T-2,\cdots,1$

$$i_t^*=\Psi_{t+1}(i_{t+1}^*)$$

求得最优路径 $I^*=(i_1^*,i_2^*,\cdots,i_T^*)$。 ■

主要代码

中文标注任务：

1、标签集合设置：

```
#33种标签
label = ['O','B-NAME', 'M-NAME', 'E-NAME', 'S-NAME','B-CONT',
         'M-CONT', 'E-CONT', 'S-CONT','B-EDU', 'M-EDU', 'E-EDU',
         'S-EDU','B-TITLE', 'M-TITLE', 'E-TITLE', 'S-TITLE',
         'B-ORG', 'M-ORG', 'E-ORG', 'S-ORG','B-RACE', 'M-RACE',
         'E-RACE', 'S-RACE','B-PRO', 'M-PRO', 'E-PRO', 'S-PRO',
         'B-LOC', 'M-LOC', 'E-LOC', 'S-LOC']
label_E = ["O","B-PER","I-PER","B-ORG","I-ORG","B-LOC","I-LOC","B-MISC","I-
MISC"]

label2num = dict()      # 标签到数字映射，用于参数学习过程建立三个参数
num2label = dict()      # 数字到标签映射，用于参数学习过程建立三个参数
num = 0
for item in label:
    label2num[item] = num
    num2label[num] = item
    num = num+1
```

2、工具函数freq2prob，用于将统计的状态转换频数、发射频数归一化，转换成概率值：

```
#将频次计数转换成概率分布
def freq2prob(d):
    '''
    输入一个频次字典，输出一个概率字典
    '''
    prob_dist = {}
    sum_freq = sum(d.values())
    for p,freq in d.items():
        if sum_freq != 0:
            prob_dist[p] = freq/sum_freq
    return prob_dist
```

3、HMM 类，类的初始化函数中包含了类的参数：标签（状态）数量、字符（观测）数量、初始概率分布`self.pi`、状态转移概率分布`self.a`、观测概率分布`self.b`等。中文任务中观测值数量取65535，即unicode编码对应的所有字符，这样等于囊括了所有的汉字和字符等，每个汉字、字符对应一个数字，比较方便。

setup函数用于设置HMM类的参数，主要是初始概率分布`self.pi`、状态转移概率分布`self.a`、观测概率分布`self.b`的学习。

`viterbi`函数用于实现维特比解码，对于输入的观测序列（中英文句子），通过维特比算法进行词性（状态）标注。

```
class HMM:
    def __init__(self,train_sents):
        self.label_num = 33
        self.char_num = 65535
        self.epsilon = 1e-50   # 无穷小量，防止归一化时分母为0
        self.a = None      # 状态转换概率矩阵
        self.b = None      # 发射概率矩阵
        self.pi = None     # 初始状态概率矩阵
        self.V = None      # 观测序列
        self.voc = set()
        self.word2num = dict()
        self.num2word = dict()
        self.setup(train_sents)

    def setup(self,train_sents):
        vocab = defaultdict(int)   # 词-词频字典
        pos = set()   # 词性集合
        for s in train_sents:
            for w, p in s:
                vocab[w] += 1
                pos.add(p)
        self.a = np.zeros((self.label_num,self.label_num))     # 状态转换概率矩阵
        self.b = np.zeros((self.label_num,self.char_num))    # 发射概率矩阵
        self.pi = np.zeros(self.label_num)    # 初始状态概率矩阵
        self.a += self.epsilon
        self.b += self.epsilon
        self.pi += self.epsilon
```

```python
        pi_freq = defaultdict(int)
        transition_freq = {}
        emission_freq = {}
        for l in label:
            transition_freq[l] = defaultdict(int)
            emission_freq[l] = defaultdict(int)
        for sent in train_sents:
            pi_freq[sent[0][1]] += 1    #统计句子开头各个状态数量
            # 记录训练集中的状态转移
            states_transition = [(p1[1],p2[1]) for p1,p2 in zip(sent,sent[1:])]

            for p1,p2 in states_transition:
                transition_freq[p1][p2] += 1
            #发射概率统计
            for w,p in sent:
                emission_freq[p][w] += 1
                self.pos.add(p)
            for p1 in label:
                for p2 in label:
                    if p2 not in transition_freq[p1]:
                        transition_freq[p1][p2] = 0
            for p1 in label:
                for p2 in self.voc:
                    emission_freq[p1][p2] = 0
        pi_freq = freq2prob(pi_freq)
        transition = {}
        for p, freq_dis in transition_freq.items():
            transition[p] = freq2prob(freq_dis)
        emission = {}
        for p, freq_dis in emission_freq.items():
            emission[p] = freq2prob(freq_dis)
        for p,q in pi_freq.items():
            self.pi[label2num[p]] = q
        for p,q in transition.items():
            for s,t in q.items():
                self.a[label2num[p],label2num[s]] = t
        for p,q in emission.items():
            for s,t in q.items():
                self.b[label2num[p], ord(s)] = t
    def viterbi(self,V):
        '''
        输入观测序列
        输出概率最大的状态序列
        :param V:
        :param a:
        :param b:
        :param initial_distribution:
        :return:
        '''
        T = V.shape[0]
        M = self.a.shape[0]

        omega = np.zeros((T, M))
        omega[0, :] = np.log(self.pi * self.b[:, V[0]])
        prev = np.zeros((T - 1, M))

        for t in range(1, T):
            for j in range(M):
```

```
                # Same as Forward Probability
                probability = omega[t - 1] + np.log(self.a[:, j]) +
np.log(self.b[j, V[t]])

                # This is our most probable state given previous state at time t
(1)
                prev[t - 1, j] = np.argmax(probability)

                # This is the probability of the most probable state (2)
                omega[t, j] = np.max(probability)

        # Path Array
        S = np.zeros(T)

        # Find the most probable last hidden state
        last_state = np.argmax(omega[T - 1, :])

        S[0] = last_state

        backtrack_index = 1
        for i in range(T - 2, -1, -1):
            S[backtrack_index] = prev[i, int(last_state)]
            last_state = prev[i, int(last_state)]
            backtrack_index += 1

        # Flip the path array since we were backtracking
        S = np.flip(S, axis=0)

        # Convert numeric values to actual hidden states
        result = []
        for s in S:
            result.append(num2label[s])
        return result
```

英文标注任务：

英文标注任务大体上和中文是相同的，但在一些处理上有差别，比如英文的观测数量，不能像中文一样简单取65535，英文中文标注的单位是汉字，而英文是单词，因此我们需要事先统计在训练集和验证集出现过的单词，制成词集当作观测的集合，做法是在主函数中先统计训练集和验证集中出现过的单词，然后作为参数传入，在setup函数中根据传入的词集确定观测数量`self.char_num`，HMM三个参数的学习、维特比解码部分与中文相同，英文任务的`HMM`类如下（仅展示不同部分）：

```
class HMM:
    def __init__(self,train_sents,amount,vocabu):
        self.label_num = 9
        self.char_num = amount
        self.epsilon = 1e-50   # 无穷小量，防止归一化时分母为0
        self.a = None     # 状态转换概率矩阵
        self.b = None     # 发射概率矩阵
        self.pi = None    # 初始状态概率矩阵
        self.V = None     # 观测序列
        self.voc = vocabu # 出现过的词的集合
        self.word2num = dict()
        self.num2word = dict()
        self.setup(train_sents)
```

```python
    def setup(self,train_sents):
        n = 0
        for v in self.voc:
            self.word2num[v] = n
            self.num2word[n] = v
            n += 1
        self.a = np.zeros((self.label_num,self.label_num))    # 状态转换概率矩阵
        self.b = np.zeros((self.label_num,self.char_num))    # 发射概率矩阵
        self.pi = np.zeros(self.label_num)    # 初始状态概率矩阵
        self.a += self.epsilon
        self.b += self.epsilon
        self.pi += self.epsilon
        print("矩阵初始化完成")

        pi_freq = defaultdict(int)
        transition_freq = {}
        emission_freq = {}
        for l in label_E:
            transition_freq[l] = defaultdict(int)
            emission_freq[l] = defaultdict(int)
        print("定义字典")
        for sent in train_sents:
            pi_freq[sent[0][1]] += 1   #统计句子开头各个状态数量
            # 记录训练集中的状态转移
            states_transition = [(p1[1],p2[1]) for p1,p2 in zip(sent,sent[1:])]
            # print("状态转移记录完成")
            for p1,p2 in states_transition:
                transition_freq[p1][p2] += 1
            for w,p in sent:
                emission_freq[p][w] += 1
        pi_freq = freq2prob(pi_freq)
        transition = {}
        for p, freq_dis in transition_freq.items():
            transition[p] = freq2prob(freq_dis)
        emission = {}
        for p, freq_dis in emission_freq.items():
            emission[p] = freq2prob(freq_dis)
        for p,q in pi_freq.items():
            self.pi[label2num[p]] = q
        for p,q in transition.items():
            for s,t in q.items():
                self.a[label2num[p],label2num[s]] = t
        for p,q in emission.items():
            for s,t in q.items():
                self.b[label2num[p],self.word2num[s]] = t
```

主函数（用于导入训练数据进行训练，并进行验证集中文标注）：

```python
if __name__=="__main__":
    load = DataLoad()
    filename = './Project2/NER/Chinese/train.txt'
    train_sents = load.load(filename)    # 中文训练集
    hmm = HMM(train_sents)
    filename2 = './Project2/NER/Chinese/validation.txt'
```

```python
    valid_sents = load.load(filename2)
    valid = []
    rigt_label = []
    for sent in valid_sents:
        s = []
        for w,l in sent:
            s.append(ord(w))
            rigt_label.append(l)
        s = np.array(s)
        valid.append(s)
    label_predict = []
    for v in valid:
        label_predict.append(hmm.viterbi(v))
        print(hmm.viterbi(v))
    num = len(rigt_label)
    # valid_sents是读取的要预测的验证集
    # label_predict是预测出来的标签

    ## 以下用于讲预测结果按验证集形式以txt文件保存，用于之后的check检查
    txt = []
    length = len(valid_sents)
    for i in range(length):
        s = []
        length2 = len(valid_sents[i])
        for j in range(length2):
            text = valid_sents[i][j][0]
            label = label_predict[i][j]
            s.append((text,label))
        txt.append(s)
    with open('myans.txt','w',encoding="utf-8") as f:
        for i in range(length):
            length2 = len(txt[i])
            for j in range(length2):
                f.write(str(txt[i][j][0])+" "+str(txt[i][j][1])+'\n')
            f.write('\n')
        f.close()
```

预测结果

中文验证集： `f1-score`达到0.8734：

```
   micro avg     0.8614    0.8856    0.8734      8437
   macro avg     0.6073    0.6577    0.6253      8437
weighted avg     0.8647    0.8856    0.8746      8437
```

英文验证集： `f1-score`达到0.8173：

```
   micro avg     0.9025    0.7468    0.8173      8603
   macro avg     0.8912    0.7311    0.8016      8603
weighted avg     0.9054    0.7468    0.8165      8603
```

完整代码

中文：

```
import numpy as np
from load import DataLoad
from collections import defaultdict
import warnings
warnings.filterwarnings("ignore")
#33种标签
label = ['O','B-NAME', 'M-NAME', 'E-NAME', 'S-NAME','B-CONT',
         'M-CONT', 'E-CONT', 'S-CONT','B-EDU', 'M-EDU', 'E-EDU',
         'S-EDU','B-TITLE', 'M-TITLE', 'E-TITLE', 'S-TITLE',
         'B-ORG', 'M-ORG', 'E-ORG', 'S-ORG','B-RACE', 'M-RACE',
         'E-RACE', 'S-RACE','B-PRO', 'M-PRO', 'E-PRO', 'S-PRO',
         'B-LOC', 'M-LOC', 'E-LOC', 'S-LOC']
label_E = ["O","B-PER","I-PER","B-ORG","I-ORG","B-LOC","I-LOC","B-MISC","I-
MISC"]

label2num = dict()     # 标签到数字映射
num2label = dict()     # 数字到标签映射
num = 0
for item in label:
    label2num[item] = num
    num2label[num] = item
    num = num+1

#将频次计数转换成概率分布
def freq2prob(d):
    '''
    输入一个频次字典，输出一个概率字典
    '''
    prob_dist = {}
    sum_freq = sum(d.values())
    for p,freq in d.items():
        if sum_freq != 0:
            prob_dist[p] = freq/sum_freq
    return prob_dist

class HMM:
    def __init__(self,train_sents):
        self.label_num = 33
        self.char_num = 65535
        self.epsilon = 1e-50  # 无穷小量，防止归一化时分母为0
        self.a = None     # 状态转换概率矩阵
        self.b = None     # 发射概率矩阵
        self.pi = None    # 初始状态概率矩阵
        self.V = None     # 观测序列
```

```python
        self.voc = set()
        self.word2num = dict()
        self.num2word = dict()
        self.setup(train_sents)

    def setup(self,train_sents):
        vocab = defaultdict(int)  # 词-词频字典
        pos = set()  # 词性集合
        for s in train_sents:
            for w, p in s:
                vocab[w] += 1
                pos.add(p)
        self.a = np.zeros((self.label_num,self.label_num))    # 状态转换概率矩阵
        self.b = np.zeros((self.label_num,self.char_num))    # 发射概率矩阵
        self.pi = np.zeros(self.label_num)   # 初始状态概率矩阵
        self.a += self.epsilon
        self.b += self.epsilon
        self.pi += self.epsilon

        pi_freq = defaultdict(int)
        transition_freq = {}
        emission_freq = {}
        for l in label:
            transition_freq[l] = defaultdict(int)
            emission_freq[l] = defaultdict(int)
        for sent in train_sents:
            pi_freq[sent[0][1]] += 1   #统计句子开头各个状态数量
            # 记录训练集中的状态转移
            states_transition = [(p1[1],p2[1]) for p1,p2 in zip(sent,sent[1:])]

            for p1,p2 in states_transition:
                transition_freq[p1][p2] += 1
            #发射概率统计
            for w,p in sent:
                emission_freq[p][w] += 1
                self.pos.add(p)
            for p1 in label:
                for p2 in label:
                    if p2 not in transition_freq[p1]:
                        transition_freq[p1][p2] = 0
            for p1 in label:
                for p2 in self.voc:
                    emission_freq[p1][p2] = 0
        pi_freq = freq2prob(pi_freq)
        transition = {}
        for p, freq_dis in transition_freq.items():
            transition[p] = freq2prob(freq_dis)
        emission = {}
        for p, freq_dis in emission_freq.items():
            emission[p] = freq2prob(freq_dis)
        for p,q in pi_freq.items():
            self.pi[label2num[p]] = q
        for p,q in transition.items():
            for s,t in q.items():
                self.a[label2num[p],label2num[s]] = t
        for p,q in emission.items():
            for s,t in q.items():
                self.b[label2num[p], ord(s)] = t
```

```
    def viterbi(self,V):
        '''
        输入观测序列
        输出概率最大的状态序列
        :param V:
        :param a:
        :param b:
        :param initial_distribution:
        :return:
        '''
        T = V.shape[0]
        M = self.a.shape[0]

        omega = np.zeros((T, M))
        omega[0, :] = np.log(self.pi * self.b[:, V[0]])
        prev = np.zeros((T - 1, M))

        for t in range(1, T):
            for j in range(M):
                # Same as Forward Probability
                probability = omega[t - 1] + np.log(self.a[:, j]) +
np.log(self.b[j, V[t]])

                # This is our most probable state given previous state at time t
(1)
                prev[t - 1, j] = np.argmax(probability)

                # This is the probability of the most probable state (2)
                omega[t, j] = np.max(probability)

        # Path Array
        S = np.zeros(T)

        # Find the most probable last hidden state
        last_state = np.argmax(omega[T - 1, :])

        S[0] = last_state

        backtrack_index = 1
        for i in range(T - 2, -1, -1):
            S[backtrack_index] = prev[i, int(last_state)]
            last_state = prev[i, int(last_state)]
            backtrack_index += 1

        # Flip the path array since we were backtracking
        S = np.flip(S, axis=0)

        # Convert numeric values to actual hidden states
        result = []
        for s in S:
            result.append(num2label[s])
        return result

if __name__=="__main__":
    load = DataLoad()
    filename = './Project2/NER/Chinese/train.txt'
    train_sents = load.load(filename)    # 中文训练集
    hmm = HMM(train_sents)
```

```python
    filename2 = './Project2/NER/Chinese/validation.txt'
    valid_sents = load.load(filename2)
    valid = []
    rigt_label = []
    for sent in valid_sents:
        s = []
        for w,l in sent:
            s.append(ord(w))
            rigt_label.append(l)
        s = np.array(s)
        valid.append(s)
    label_predict = []
    for v in valid:
        label_predict.append(hmm.viterbi(v))
        print(hmm.viterbi(v))
    num = len(rigt_label)
    # valid_sents是读取的要预测的验证集
    # label_predict是预测出来的标签

    num2 = 0
    myans = []
    for sent in label_predict:
        num2 += len(sent)
        for s in sent:
            myans.append(s)
    print(num)
    print(num2)
    score = 0
    for i in range(len(myans)):
        if(myans[i] == rigt_label[i]):
            score += 1
    print('{}/{} = {}'.format(score,len(myans),score/len(myans)))
    print(score/len(myans),)

    ## 以下用于讲预测结果按验证集形式以txt文件保存，用于之后的check检查
    txt = []
    length = len(valid_sents)
    for i in range(length):
        s = []
        length2 = len(valid_sents[i])
        for j in range(length2):
            text = valid_sents[i][j][0]
            label = label_predict[i][j]
            s.append((text,label))
        txt.append(s)
    with open('myans.txt','w',encoding="utf-8") as f:
        for i in range(length):
            length2 = len(txt[i])
            for j in range(length2):
                f.write(str(txt[i][j][0])+" "+str(txt[i][j][1])+'\n')
            f.write('\n')
        f.close()
```

英文:

```python
import numpy as np
from load import DataLoad
```

```python
from collections import defaultdict
import warnings
warnings.filterwarnings("ignore")

label_E = ["O","B-PER","I-PER","B-ORG","I-ORG","B-LOC","I-LOC","B-MISC","I-
MISC"]

label2num = dict()
num2label = dict()
num = 0
for item in label_E:
    label2num[item] = num
    num2label[num] = item
    num = num+1

#将频次计数转换成概率分布
def freq2prob(d):
    '''
    输入一个频次字典，输出一个概率字典
    '''
    prob_dist = {}
    sum_freq = sum(d.values())
    for p,freq in d.items():
        if sum_freq != 0:
            prob_dist[p] = freq/sum_freq
    return prob_dist
class HMM:
    def __init__(self,train_sents,amount,vocabu):
        self.label_num = 9
        self.char_num = amount
        self.epsilon = 1e-50   # 无穷小量，防止归一化时分母为0
        self.a = None     # 状态转换概率矩阵
        self.b = None     # 发射概率矩阵
        self.pi = None    # 初始状态概率矩阵
        self.v = None     # 观测序列
        # self.pos = set()
        self.voc = vocabu # 出现过的词的集合
        self.word2num = dict()
        self.num2word = dict()
        self.setup(train_sents)

    def setup(self,train_sents):
        n = 0
        for v in self.voc:
            self.word2num[v] = n
            self.num2word[n] = v
            n += 1
        print("老子要看这里")
        self.a = np.zeros((self.label_num,self.label_num))     # 状态转换概率矩阵
        self.b = np.zeros((self.label_num,self.char_num))     # 发射概率矩阵
        self.pi = np.zeros(self.label_num)    # 初始状态概率矩阵
        self.a += self.epsilon
        self.b += self.epsilon
        self.pi += self.epsilon
        print("矩阵初始化完成")

        pi_freq = defaultdict(int)
        transition_freq = {}
```

```
        emission_freq = {}
        for l in label_E:
            transition_freq[l] = defaultdict(int)
            emission_freq[l] = defaultdict(int)
        print("定义字典")
        for sent in train_sents:
            pi_freq[sent[0][1]] += 1   #统计句子开头各个状态数量
            states_transition = [(p1[1],p2[1]) for p1,p2 in zip(sent,sent[1:])]
            for p1,p2 in states_transition:
                transition_freq[p1][p2] += 1
            for w,p in sent:
                emission_freq[p][w] += 1
        print("开始字典转概率")
        pi_freq = freq2prob(pi_freq)
        transition = {}
        for p, freq_dis in transition_freq.items():
            transition[p] = freq2prob(freq_dis)
        emission = {}
        for p, freq_dis in emission_freq.items():
            emission[p] = freq2prob(freq_dis)
        print("结束字典转概率")
        print("开始填三个矩阵")
        for p,q in pi_freq.items():
            self.pi[label2num[p]] = q
        for p,q in transition.items():
            for s,t in q.items():
                self.a[label2num[p],label2num[s]] = t
        for p,q in emission.items():
            for s,t in q.items():
                self.b[label2num[p],self.word2num[s]] = t
        print("设置已完成")
    def viterbi(self,V):
        '''
        输入模型和观测序列
        输出概率最大的状态序列
        :param V:
        :param a:
        :param b:
        :param initial_distribution:
        :return:
        '''
        T = V.shape[0]
        M = self.a.shape[0]

        omega = np.zeros((T, M))
        omega[0, :] = np.log(self.pi * self.b[:, V[0]])


        prev = np.zeros((T - 1, M))

        for t in range(1, T):
            for j in range(M):
                # Same as Forward Probability
                probability = omega[t - 1] + np.log(self.a[:, j]) +
np.log(self.b[j, V[t]])

                # This is our most probable state given previous state at time t
(1)
```

```
                prev[t - 1, j] = np.argmax(probability)

                # This is the probability of the most probable state (2)
                omega[t, j] = np.max(probability)

        # Path Array
        S = np.zeros(T)

        # Find the most probable last hidden state
        last_state = np.argmax(omega[T - 1, :])

        S[0] = last_state

        backtrack_index = 1
        for i in range(T - 2, -1, -1):
            S[backtrack_index] = prev[i, int(last_state)]
            last_state = prev[i, int(last_state)]
            backtrack_index += 1

        # Flip the path array since we were backtracking
        S = np.flip(S, axis=0)

        # Convert numeric values to actual hidden states
        result = []
        for s in S:
            result.append(num2label[s])
        return result

if __name__=="__main__":
    load = DataLoad()
    filename_E = './Project2/NER/English/train.txt'
    train_sents = load.load_E(filename_E) # 英文训练集
    filename2_E = './Project2/NER/English/validation.txt'
    valid_sents_E = load.load_E(filename2_E)
    vocabu = set()
    for s in train_sents:
        for w, p in s:
            vocabu.add(w)

    for s in valid_sents_E:
        for w, p in s:
            vocabu.add(w)
    # amount = len(vocabu)
    # print(amount)
    hmm = HMM(train_sents,amount,vocabu)
    valid = []
    right_label = []
    for sent in valid_sents_E:
        s = []
        for w,l in sent:
            s.append(hmm.word2num[w])
            right_label.append(l)
        s = np.array(s)
        valid.append(s)    # 预测值，按句子分
    label_predict = []
    for v in valid:
        label_predict.append(hmm.viterbi(v))
    num = len(right_label)
```

```
    txt = []
    length = len(valid_sents_E)
    for i in range(length):
        s = []
        length2 = len(valid_sents_E[i])
        for j in range(length2):
            text = valid_sents_E[i][j][0]
            label = label_predict[i][j]
            s.append((text,label))
        txt.append(s)

    with open('part1_E.txt','w',encoding="utf-8") as f:
        for i in range(length-1):
            length2 = len(txt[i])
            for j in range(length2):
                f.write(str(txt[i][j][0])+" "+str(txt[i][j][1])+'\n')
            f.write('\n')
        length2 = len(txt[length - 1])
        for j in range(length2):
            f.write(str(txt[length - 1][j][0]) + " " + str(txt[length - 1][j]
[1]) + '\n')
        f.close()
```