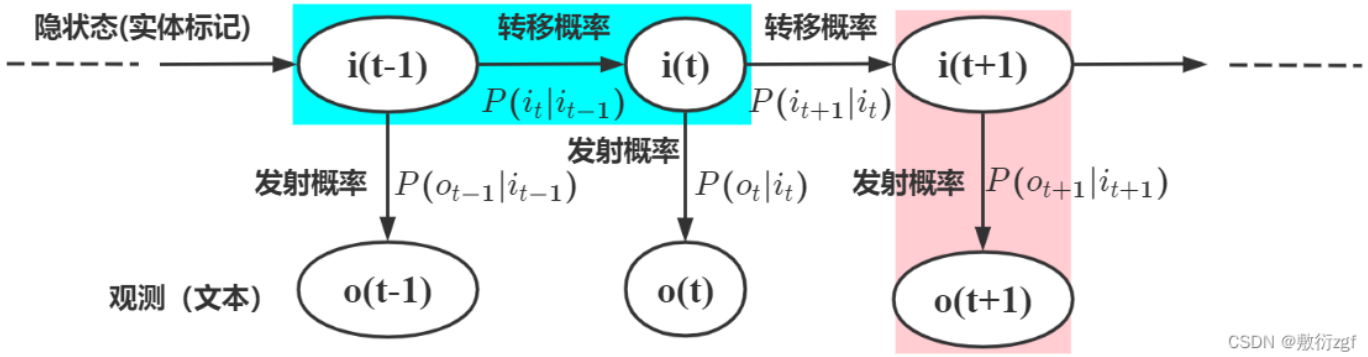
pj2 part1 HMM模型

实验原理



1. 第 t 个隐状态(实体标签)只跟前一时刻的 t—1 隐状态(实体标签)有关, 与除此之外的其他隐状态无关.

2. 观测独立的假设, HMM模型中是由隐状态序列(实体标记)生成可观测状态(可读文本)的过程, 观测独立假设是指在任意时刻观测 ot 只依赖于当前时刻的隐状态 it , 与其他时刻的隐状态无关.

HMM的原理较为简单，其实现过程最重要的是初始矩阵pi，转移矩阵A，发射矩阵B的计算和使用。由于标签数较多，在确定最有路径时使用viterbi解码的方法可以将复杂度从o(tag^n)降低到o(tag).

Code

load Plain Text 复制代码

```
label_classes = ['NAME', 'CONT', 'EDU', 'TITLE', 'ORG', 'RACE', 'PRO', 'LOC']

tag2idx = defaultdict()
tag2idx['O'] = 0
count = 1
for label in label_classes:
    tag2idx['B-' + label] = count
    count += 1
    tag2idx['M-' + label] = count
    count += 1
    tag2idx['E-' + label] = count
    count += 1
    tag2idx['S-' + label] = count
    count += 1

def load_data(data_path: str):
    sentences = []
    with open(data_path, 'r', encoding='utf-8') as fp:
        sentence = []
        text_data = []
        label_data = []
        for line in fp:
            if len(line) > 1:
                text = line[0]
                label = line[2:-1]
                text_data.append(text)
                label_data.append(label)
            else:
                sentence.append(text_data)
                sentence.append(label_data)
                sentences.append(sentence)
                sentence = []
                text_data = []
                label_data = []
    return sentences

sentences = load_data("./Chinese/train.txt")
```

以中文NER为例，首先生成数据集。通过对于txt文件的操作，可以产生一个标签字典和一个以字作为元素、句子作为单位的列表。

Getdict Plain Text 复制代码

```
def GetDict(path_lists):
    word_dict = OrderedDict()
    for path in path_lists:
        with open(path, "r", encoding="utf-8") as f:
            annotations = f.readlines()
        for annotation in annotations:
            splited_string = annotation.strip(" ").split(" ")
            if len(splited_string)<=1:
                continue
            word = splited_string[0]
            if word not in word_dict:
                word_dict[word] = len(word_dict)
    return word_dict
```

之前讲过的字典是为了将标签数字化，同样的我们也需要将数据集中的每个具体文字数据化。这里的处理方式在中文和英文中有些许不同：中文的数据以字为单位，所以直接使用汉字对应的码数ord(char)作为标号；英文的数据以单词为单位，无法直接对于一整个单词编码，故我们额外写一个字典，将训练集和测试集中出现的每个单词写入字典。

class HMM_model

Plain Text | 复制代码

```
class HMM_model:
    def __init__(self, tag2idx):
        self.tag2idx = tag2idx  # tag2idx字典
        self.n_tag = len(self.tag2idx)  # 标签个数
        self.n_char = 65535  # 所有字符的Unicode编码个数，包括汉字
        self.epsilon = 1e-100  # 无穷小量，防止归一化时分母为0
        self.idx2tag = dict(zip(self.tag2idx.values(), self.tag2idx.keys
()))  # idx2tag字典
        self.A = np.zeros((self.n_tag, self.n_tag))  # 状态转移概率矩阵，sha
pe:(21, 21)
        self.B = np.zeros((self.n_tag, self.n_char))  # 观测概率矩阵，shape:
(21, 65535)
        self.pi = np.zeros(self.n_tag)  # 初始隐状态概率,shape：(21,)

    def train(self, train_data):
        print('开始训练数据：')
        for i in tqdm(range(len(train_data))):  # 几组数据
            for j in range(len(train_data[i][0])):  # 每组数据中几个字符
                cur_char = train_data[i][0][j]  # 取出当前字符
                cur_tag = train_data[i][1][j]  # 取出当前标签
                self.B[self.tag2idx[cur_tag]][ord(cur_char)] += 1  # 对B矩
阵中标签->字符的位置加一
                if j == 0:
                    # 若是文本段的第一个字符，统计pi矩阵
                    self.pi[self.tag2idx[cur_tag]] += 1
                    continue
                pre_tag = train_data[i][1][j - 1]  # 记录前一个字符的标签
                self.A[self.tag2idx[pre_tag]][self.tag2idx[cur_tag]] += 1
 # 对A矩阵中前一个标签->当前标签的位置加一

        # 防止数据下溢,对数据进行对数归一化
        self.A[self.A == 0] = self.epsilon
        self.A = np.log(self.A) - np.log(np.sum(self.A, axis=1, keepdims=T
rue))
        self.B[self.B == 0] = self.epsilon
        self.B = np.log(self.B) - np.log(np.sum(self.B, axis=1, keepdims=T
rue))
        self.pi[self.pi == 0] = self.epsilon
        self.pi = np.log(self.pi) - np.log(np.sum(self.pi))

        # 将A，B，pi矩阵保存到本地
        np.savetxt('./Chinese/A.txt', self.A)
        np.savetxt('./Chinese/B.txt', self.B)
        np.savetxt('./Chinese/pi.txt', self.pi)
        print('训练完毕！')
```

```
39
40
41      # 载入A，B，pi矩阵参数
        def load_paramters(self, A='./Chinese/A.txt', B='./Chinese/B.txt', pi
    ='./Chinese/pi.txt'):
42          self.A = np.loadtxt(A)
43          self.B = np.loadtxt(B)
44          self.pi = np.loadtxt(pi)
45
46      # 使用维特比算法进行解码
47      def viterbi(self, s):
48          # 计算初始概率，pi矩阵+第一个字符对应各标签概率
49          delta = self.pi + self.B[:, ord(s[0])]
50          # 前向传播记录路径
51          path = []
52          for i in range(1, len(s)):
53              # 广播机制，重复加到A矩阵每一列
54              tmp = delta.reshape(-1, 1) + self.A
55              # 取最大值作为节点值，并加上B矩阵
56              delta = np.max(tmp, axis=0) + self.B[:, ord(s[i])]
57              # 记录当前层每一个节点的最大值来自前一层哪个节点
58              path.append(np.argmax(tmp, axis=0))
59
60          # 回溯，先找到最后一层概率最大的索引
61          index = np.argmax(delta)
62          results = [self.idx2tag[index]]
63          # 逐层回溯，沿着path找到起点
64          while path:
65              tmp = path.pop()
66              index = tmp[index]
67              results.append(self.idx2tag[index])
68          # 序列翻转
69          results.reverse()
70          return results
71
72      def predict(self, s):
73          results = self.viterbi(s)
74          for i in range(len(s)):
75              print(s[i] + results[i], end=' | ')
76
77      def valid(self, valid_data):
78          y_pred = []
79          # 遍历验证集每一条数据，使用维特比算法得到预测序列，并加到列表中
80          for i in range(len(valid_data)):
81              y_pred.append(self.viterbi(valid_data[i][0]))
82          return y_pred
```

这段代码是HMM模型类的定义，现在挑选其中重要函数进行阐述：

1. train函数：HMM中的训练过程其实只是一个统计过程。遍历所有数据，对于pi初始矩阵来说，若是文本段的第一个字符，统计pi矩阵；对于发射矩阵B来说，在遍历过程中对B矩阵中标签–>字符的位置加1；对于转移矩阵A来说，对A矩阵中前一个标签–>当前标签的位置加1。遍历结束后，归一化操作中只要确保分母不为0即可，将矩阵中的0都替换成无穷小量。
2. viterbi解码函数：维特比解码分为前溯过程和回溯过程。在前溯过程中，先计算初始概率，求出pi矩阵+第一个字符对应各标签概率，然后遍历每一列，记录当前层每一个节点的最大值来自前一层哪个节点，最后一共得到的可能路径有tag条（因为最后一列的每一个tag只有唯一最有来路）；在回溯过程中，先找到最后一层概率最大的索引，然后逐层回溯，沿着path找到起点，最后通过一次序列翻转得到标签路径。

实验结果

中文validation上的精度：

	precision	recall	f1-score	support
B-NAME	0.8835	0.8922	0.8878	102
M-NAME	0.8310	0.7867	0.8082	75
E-NAME	0.7723	0.7647	0.7685	102
S-NAME	0.5000	0.7500	0.6000	8
B-CONT	0.9706	1.0000	0.9851	33
M-CONT	0.9846	1.0000	0.9922	64
E-ORG	0.7618	0.7720	0.7669	522
S-ORG	0.0000	0.0000	0.0000	0
B-RACE	1.0000	1.0000	1.0000	14
M-RACE	0.0000	0.0000	0.0000	0
E-RACE	1.0000	1.0000	1.0000	14
S-RACE	0.2500	1.0000	0.4000	1
B-PRO	0.3714	0.7222	0.4906	18
M-PRO	0.2982	0.5152	0.3778	33
E-PRO	0.4857	0.9444	0.6415	18
S-PRO	0.0000	0.0000	0.0000	0
B-LOC	0.3333	0.5000	0.4000	2
M-LOC	0.7500	0.5000	0.6000	6
E-LOC	0.3333	0.5000	0.4000	2
S-LOC	0.0000	0.0000	0.0000	0
micro avg	0.8451	0.8747	0.8596	8437
macro avg	0.5817	0.6543	0.6033	8437
weighted avg	0.8695	0.8747	0.8716	8437

英文validation上的精度：

```
ER/check.py
              precision    recall  f1-score   support

       B-PER     0.9453    0.7041    0.8071      1842
       I-PER     0.8915    0.7988    0.8426      1307
       B-ORG     0.6654    0.7860    0.7207      1341
       I-ORG     0.6347    0.7217    0.6754       751
       B-LOC     0.9035    0.8253    0.8626      1837
       I-LOC     0.4515    0.7432    0.5618       257
      B-MISC     0.5409    0.8254    0.6535       922
      I-MISC     0.3063    0.7197    0.4297       346

   micro avg     0.7153    0.7735    0.7433      8603
   macro avg     0.6674    0.7655    0.6942      8603
weighted avg     0.7737    0.7735    0.7604      8603
```

实验思考

HMM模型的序列标注精度不算很高。根据其原理可能的解释如下：

1. 特征表示局限性：HMM模型在序列标注任务中特征仅依赖相邻文本，可能不足以捕捉语言的细粒度特征，忽略了更长的上下文信息。
2. 全局依赖建模困难：HMM模型的状态转移概率是局部计算的，无法考虑跨越多个状态的全局依赖关系。某些状态的标注可能取决于整个序列的结构，此时HMM模型的精度可能不足以满足需求。

参考链接

https://github.com/ZejunCao/NER_baseline/blob/main/HMM.py