模式识别第一次实验报告

162050127 颜劭铭

2022 年 12 月 15 日

1 问题场景

本次实验将通过 pytorch 作为主要工具实现 HMM 来解决中文命名实体识别问题。

首先，明确一下命名实体识别的概念：

命名实体识别（Named Entity Recognition），简称 NER，是指识别文本中具有特定意义的实体，是信息提取、问答系统、句法分析、机器翻译等应用领域的重要基础工具，在自然语言处理技术走向实用化的过程中占有重要地位。一般来说，命名实体识别的任务就是识别出待处理文本中三大类（实体类、时间类和数字类）、七小类（人名、机构名、地名、时间、日期、货币和百分比）命名实体。

假如有这么一句话：小明早上 8 点去学校上课。

对其进行命名实体识别，应该能提取信息如下：

- 人名：小明
- 时间：早上 8 点
- 地点：学校

2 数据集

数据集用的是论文 ACL 2018Chinese NER using Lattice LSTM 中从新浪财经收集的简历数据, 它的每一行由一个字及其对应的标注组成，标注集采用 BIOES 标注方案，句子之间用一个空行隔开。

BIOES 标注方案：

- B 表示 Begin，命名实体的第一个字
- I 表示 Inside，命名实体中除了第一个字和最后一个字以外的其他字
- O 表示 Outside，不属于命名实体
- E 表示 End，命名实体的最后一个字
- S 表示 Single，仅由一个字组成的命名实体

数据的格式如下所示:

- 美 B-LOC
- 国 E-LOC
- 的 O
- 华 B-PER

- 莱 I-PER
- 士 E-PER
- 我 O
- 跟 O
- 他 O
- 谈 O
- 笑 O
- 风 O
- 生 O

其中 PER 指的是任务，LOC 指的是地名。

3 软件包

主要使用了 PyTorch

PyTorch 是由 Facebook 人工智能研究院的 Torch7 团队开发的开源的深度学习框架，它的底层是由 C++ 实现，但是实现和运用全部是用 Python 完成。

4 实验思路

本次试验主要使用的方法是 HMM，即隐马尔科夫模型，主要描述的是由一个隐藏的马尔科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产生的观测随机序列的过程。马尔科夫链即在特定的情况下，该系统在时间 t 的状态只与其在时间 t-1 的状态相关，则该系统被称为一个离散的一阶马尔科夫链。隐马尔科夫模型主要包括状态数，每个状态可能的符号数（加起来就是观测数），状态转移矩阵，从某个状态观察到某一特定符号观测概率矩阵，初始状态的概率分布五个要素。

而 NER 本质上是一种序列标注问题，即预测每个字的 BIOES 标记，而使用 HMM 解决 NER 的问题时，对应的五个要素的解释为：

1) 状态数：即每个字对应的标注，也就是对应存在的标注的种类，是观测不到的
2) 观测数，即存在有多少不同的字，也就是我们观测到的字组成的序列
3) 状态转移概率矩阵就是由某一个标注转移到下一个标注的概率（设状态转移矩阵为 M，那么若前一个词的标注为 tag_i，则下一个词的标注为 tag_j 的概率为 M_{ij}）
4) 观测概率矩阵就是指在某个标注下，生成某个词的概率
5) 初始状态分布就是每一个标注作为句子第一个字的标注的概率

因此根据五个要素，可以定义如下的 HMM 模型：

```
class HMM(object):
    def __init__(self, N, M):
        """Args:
            N: 状态数，这里对应存在的标注的种类
            M: 观测数，这里对应有多少不同的字
```

```
        """
        self.N = N
        self.M = M

        # 状态转移概率矩阵 A[i][j]表示从i状态转移到j状态的概率
        self.A = torch.zeros(N, N)
        # 观测概率矩阵, B[i][j]表示i状态下生成j观测的概率
        self.B = torch.zeros(N, M)
        # 初始状态概率  Pi[i]表示初始时刻为状态i的概率
        self.Pi = torch.zeros(N)
```

HMM 模型的训练过程实际上就是根据训练数据和最大似然估计方法估计模型的三个要素，即上文提到的初始状态分布、状态转移概率矩阵以及观测概率矩阵（状态数和观测数已知）。比如说，在估计初始状态分布的时候，假如某个标记在数据集中作为句子第一个字的标记的次数为 k，句子的总数为 N，那么该标记作为句子第一个字的概率可以近似估计为 k/N。代码如下所示：

```
    def train(self, word_lists, tag_lists, word2id, tag2id):
        """HMM的训练，即根据训练语料对模型参数进行估计,
        因为我们有观测序列以及其对应的状态序列，所以我们
        可以使用极大似然估计的方法来估计隐马尔可夫模型的参数
        参数:
            word_lists: 列表，其中每个元素由字组成的列表，如 ['担','任','科','员']
            tag_lists: 列表，其中每个元素是由对应的标注组成的列表，如 ['O','O','B-TITLE', 'E-TITLE']
            word2id: 将字映射为ID
            tag2id: 字典，将标注映射为ID
        """

        assert len(tag_lists) == len(word_lists)

        # 估计转移概率矩阵
        for tag_list in tag_lists:
            seq_len = len(tag_list)
            for i in range(seq_len - 1):
                current_tagid = tag2id[tag_list[i]]
                next_tagid = tag2id[tag_list[i+1]]
                self.A[current_tagid][next_tagid] += 1
        # 问题：如果某元素没有出现过，该位置为0，这在后续的计算中是不允许的
        # 解决方法：我们将等于0的概率加上很小的数
        self.A[self.A == 0.] = 1e-10
        self.A = self.A / self.A.sum(dim=1, keepdim=True)

        # 估计观测概率矩阵
        for tag_list, word_list in zip(tag_lists, word_lists):
            assert len(tag_list) == len(word_list)
            for tag, word in zip(tag_list, word_list):
                tag_id = tag2id[tag]
                word_id = word2id[word]
                self.B[tag_id][word_id] += 1
        self.B[self.B == 0.] = 1e-10
        self.B = self.B / self.B.sum(dim=1, keepdim=True)

        # 估计初始状态概率
        for tag_list in tag_lists:
            init_tagid = tag2id[tag_list[0]]
            self.Pi[init_tagid] += 1
        self.Pi[self.Pi == 0.] = 1e-10
        self.Pi = self.Pi / self.Pi.sum()
```

模型训练完毕之后，要利用训练好的模型进行解码，就是对给定的模型未见过的句子，求句子中的每个字对应的标注，针对这个解码问题，使用的是维特比（viterbi）算法。实际是用动态规划解隐马尔可夫

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。

5 效果展示

训练并测试结果如图1所示，第一列是不同标注状态名称，第二列是准确率，第三列是召回率，第四列是 F1 分数，第五列是数据集中该状态的总数，最后一行则是在所有标注中的平均结果，可以看到 HMM 的训练非常快，仅仅需要 14 秒左右，而准确率也可以达到 91.49，相比较原论文作者提出的 LSTM 模型的准确度为 94.81，可以发现还是不错的。

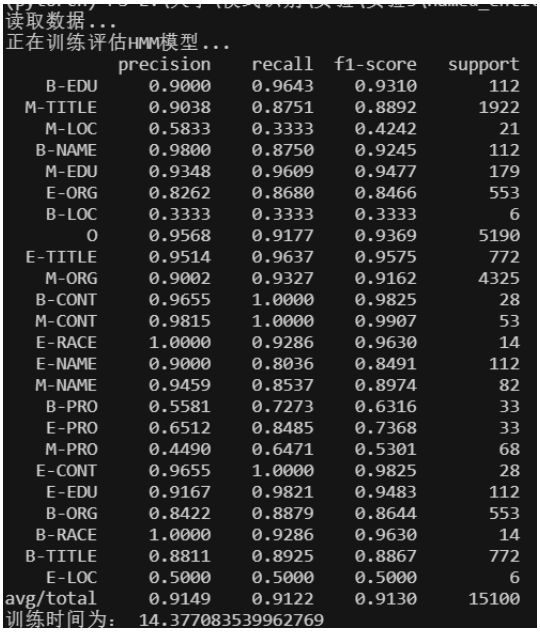

图 1: 训练并测试结果图