HMM 实验报告

1. 实验原理

序列 $s = a_1 a_2 \dots a_n$，其对应的词性为 $= l_1, l_2 \dots l_n$。

已知序列集 $S = \{s_1, s_2, \dots, s_m\}$，对应的词性集合 $L = \{l_1, l_2, \dots, l_m\}$，其中词表的大小为 M，词性集大小为N；

得出状态转移矩阵 $A_{N\times N}$ 、观测状态矩阵 $B_{M\times N}$ 和初始状态分布向量 Π_N 。

其中 A_{ij} 表示从词性 i 转移到词性 j 的概率；B_{ij} 表示单词 i 的词性为 j 的概率；Π_i 表示词性为 i 的概率

输入一序列 s，其长度为 T, 要求其对应的词性序列 l，可通过维特比算法求解：

1. 初始化局部状态：

$$\delta_1(i) = \pi_i b_i\left(o_1\right), i = 1, 2 \dots N$$
$$\Psi_1(i) = 0, i = 1, 2 \dots N$$

2. 递推时刻 $t = 2, 3, \dots, T$ 时刻的局部状态:

$$\delta_t(i) = \max_{1\le j\le N}\left[\delta_{t-1}(j) a_{ji}\right] b_i\left(0_t\right), i = 1, 2 \dots N$$
$$\Psi_t(i) = \arg\max_{1\le j\le N}\left[\delta_{t-1}(j) a_{ji}\right], i = 1, 2 \dots N$$

3. 计算时刻 T 最大的 $\Psi_t(i)$, 即为时刻 T 最可能的隐藏状态:

$$P_* = \max_{1\le j\le N} \delta_T(i)$$
$$i_T^* = \arg\max_{1\le j\le N}\left[\delta_T(i)\right]$$

4. 利用局部状态 $\Psi_t(i)$ 开始回溯，对于 $t = T-1, T-2, \dots, 1$:

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

- 最终得到最有可能的词性序列 $I^* = \{i_1^*, i_2^*, \dots i_T^*\}$

2. 实验过程

数据处理：

1. 修正词性集外的错误标注，nhf 和 nhs 修正为 nh，mq 修正为 m
2. 读取数据集，得到单词序列集合和对应的词性集合
3. 统计词表，并将单词序列集合转化为对应的词表 id 序列集合；将词性集合转换为对应的标签集合
4. 按 9 : 1 随机划分训练集和测试集

关键代码：

1. 计算转移矩阵、状态矩阵和初始状态向量

```python
# 计算转移矩阵、状态矩阵和初始状态
def hmm(self, vocab_ids: List[List[int]], pos_labels: List[List[int]]):
    for sentence_ids, sentence_labels in zip(vocab_ids, pos_labels):
        for word_index in range(len(sentence_ids)):
            word_id = sentence_ids[word_index]
            pos_label = sentence_labels[word_index]
            # 记录状态加 1
            self._state[word_id][pos_label] += 1
            if word_index - 1 > 0:
                last_pos_label = sentence_labels[word_index - 1]
                # 转移状态加 1
                self._transition[last_pos_label][pos_label] += 1
    # 计算转移概率矩阵、状态矩阵和初始状态向量
    np.divide(self._state, np.sum(self._state, axis=1, keepdims=True), out=self.state_matrix)
    self.state_matrix[np.isnan(self.state_matrix)] = 0

    np.divide(self._transition, np.sum(self._transition, axis=1, keepdims=True), out=self.transition_matrix)
    self.transition_matrix[np.isnan(self.transition_matrix)] = 0

    np.divide(np.sum(self._state, axis=0), np.sum(self._state), out=self.start_state)
```

2. 利用维特比算法求词性

```python
def vetebi(self, sentence_ids):
    word_num = len(sentence_ids)
    if word_num == 0:
        return []
    # 记录两个局部状态
    delta = np.zeros((word_num, self.pos_num))
    psi = np.zeros((word_num, self.pos_num), dtype=int)

    # 初始化两个局部状态
    np.multiply(self.start_state, self.state_matrix[sentence_ids[0]], out=delta[0])

    # 计算接下来的时间状态
    for word_index in range(1, word_num):
        word_id = sentence_ids[word_index]
        for pos_id in range(self.pos_num):
            temp = delta[word_index - 1] * self.transition_matrix[:, pos_id]
            max_index = temp.argmax()
            psi[word_index][pos_id] = max_index
            delta[word_index][pos_id] = temp[max_index] * self.state_matrix[word_id][pos_id]

    # 回溯
    pos_labels = []
    pos_label = delta[-1].argmax()
    pos_labels.append(pos_label)
    for step in range(word_num - 1, 0, -1):
        pos_label = psi[step][pos_label]
        pos_labels.append(pos_label)
    pos_labels.reverse()
    return pos_labels
```

实验结果：

输出测试集正确率：

```
acc: 66.02%
```

3. 实验总结

1. 掌握了 HMM 维特比算法的求解原理
2. 本次实验利用 HMM 算法求解了词性标注问题
3. 通过本次实验，能够将该算法推广应用到实际问题中去