Project2 HMM, CRF, BiLSTM+CRF

21307140069 田沐钊

Part 1:HMM实现

HMM用于建模具有隐含状态的序列数据。它假设观测序列的生成由隐状态控制，且隐状态之间的关系满足一阶齐次马尔可夫假设。HMM由两组参数组成：状态转移概率矩阵和发射概率矩阵。HMM在序列标注问题中表现良好。通过学习观测序列和隐藏状态之间的概率分布，HMM可以对观测序列进行标注，识别和提取出隐藏状态的信息。这使得HMM在命名实体识别、词性标注等任务中有很好的应用效果。
HMM提供了一种对序列数据进行建模和解释的框架。通过观察状态转移概率矩阵和观测概率矩阵，可以了解隐藏状态之间的转移规律以及观测序列与隐藏状态之间的关联程度。这使得HMM在一定程度上具有可解释性，有助于理解和分析问题。

以下是HMM的代码实现和运行结果：

```
class HMM():
    def __init__(self, num_state, num_words):
        self.num_state = num_state
        self.num_words = num_words
        # 该初识化在本任务中并无作用，但在用EM算法进行参数估计时会有用
        self.initial_probability = np.random.rand(num_state)
        self.initial_probability /= np.sum(self.initial_probability)

        self.transition_probability = np.random.rand(num_state, num_state)
        self.transition_probability /= np.sum(self.transition_probability,
axis=1, keepdims=True)

        self.emission_probability = np.random.rand(num_state, num_words)
        self.emission_probability /= np.sum(self.emission_probability,
axis=1, keepdims=True)

    # 通过对训练集的简单统计得到初始概率、转移概率和发射概率
    def param_estimate(self, dataset, smoothing_factor=1):
        self.initial_probability = smoothing_factor *
np.ones_like(self.initial_probability)
        self.transition_probability = smoothing_factor *
np.ones_like(self.transition_probability)
        self.emission_probability = smoothing_factor *
np.ones_like(self.emission_probability)

        for words, tags in dataset:
            self.initial_probability[tags[0]] += 1
            for i in range(len(tags) - 1):
                self.transition_probability[tags[i], tags[i + 1]] += 1
                self.emission_probability[tags[i], words[i]] += 1
            self.emission_probability[tags[-1], words[-1]] += 1

        self.initial_probability /= np.sum(self.initial_probability)
        self.transition_probability /= np.sum(self.transition_probability,
axis=1, keepdims=True)
```

```
        self.emission_probability /= np.sum(self.emission_probability,
axis=1, keepdims=True)

    def evaluate(self, words):
        length = len(words)
        # prob[t,i] 为前 t 项观测值符合观测序列且第t项的状态为i的情况所对应的概率
        prob = np.zeros((length + 1, self.num_state))
        prob[0] = self.initial_probability
        for t in range(length):
            for current_state in range(self.num_state):
                for previous_state in range(self.num_state):
                    prob[t + 1, current_state] += prob[t, previous_state] *
self.transition_probability[previous_state, current_state] * \

 self.emission_probability[previous_state, words[t]]

        return np.sum(prob[-1])

    def decode(self, words):
        length = len(words)

        # 使用概率原值进行累乘，会使结果过小，导致下溢，故此处将其转换到对数空间上进行运算
        delta = np.full((length, self.num_state), -np.inf)
        psi = np.zeros((length, self.num_state), dtype=int)

        delta[0] = np.log(self.initial_probability) +
np.log(self.emission_probability[:, words[0]])
        for t in range(1, length):
            for current_state in range(self.num_state):
                prob = delta[t - 1] + np.log(self.transition_probability[:,
current_state]) + np.log(self.emission_probability[current_state, words[t]])
                delta[t, current_state] = np.max(prob)
                psi[t, current_state] = np.argmax(prob)

        states = np.zeros(length, dtype=int)
        states[-1] = np.argmax(delta[-1])
        for t in range(length - 1, 0, -1):
            states[t - 1] = psi[t, states[t]]

        return states
```

```
---------------- Chinese Result ------------------
              precision    recall  f1-score   support
   micro avg     0.8693    0.9147    0.8914      8437
   macro avg     0.5353    0.5909    0.5565      8437
weighted avg     0.8720    0.9147    0.8919      8437

---------------- English Result ------------------
              precision    recall  f1-score   support

   micro avg     0.7607    0.7238    0.7418      8603
   macro avg     0.7287    0.6968    0.7056      8603
weighted avg     0.7796    0.7238    0.7448      8603
```

Part 2: 基于sklearn_crfsuite的CRF实现：

CRF通过学习条件概率分布来建模观测序列和标签序列之间的关系，在许多自然语言处理任务中表现出色。

其基于无向图模型，将标注问题转化为在图结构上进行推断的问题。CRF假设给定观测序列的情况下，标签序列的生成过程是由一个概率图模型控制的。该模型中的节点表示观测序列和标签序列的某个位置，边表示两个位置之间的依赖关系。通过基于特征函数对每个状态转移路径进行打分来对概率进行建模。

CRF能够建模更复杂的依赖关系。与HMM不同，CRF不仅考虑当前观测和标签的关系，还考虑了上下文中其他标签的影响。这使得CRF能够更准确地捕捉到序列中的长期依赖关系，提高了建模能力。此外，CRF使用对数线性模型作为其条件概率分布，可以通过对数似然函数进行最大化来学习模型参数。与基于局部的模型（如最大熵模型）相比，CRF进行全局优化，考虑了整个标签序列的概率分布，能够更好地解决标签之间的冲突和歧义。CRF可以使用丰富的特征表示来描述观测序列和标签序列之间的关系。这些特征可以基于词性、上下文、形态等多种信息，能够灵活地捕捉到序列中的各种规律和模式。

以下是sklearn_CRF的代码实现和运行结果：

```python
import os
import sklearn_crfsuite
from check import *
from NER_dataset import *
from word2features import *
from utils import *


def CRF_check(language):
    train_data, valid_data = get_data_set(language)
    train_features, train_labels = get_features(train_data)

    # 建立并训练CRF模型
    model = sklearn_crfsuite.CRF(
        algorithm='lbfgs',
        c1=0.1,
        c2=0.1,
        max_iterations=1000,
        all_possible_transitions=True
    )
    try:
        model.fit(train_features, train_labels)
    except AttributeError:
        pass

    # 输出
    test_features, test_labels = get_features(valid_data)
    pred = model.predict(test_features)

    my_path = 'my_{language}_CRF_result.txt'
    gold_path = f'./NER/{language}/validation.txt'

    file = open(my_path, "w")
    for idx, words_tags in enumerate(valid_data):
        words, _ = words_tags
        tags = pred[idx]
```

```
        for i in range(len(words)):
            file.write(f'{words[i]} {tags[i]}\n')
        file.write('\n')
    file.close()

    check(language, gold_path, my_path)

print('---------------- Chinese Result ------------------')
CRF_check('Chinese')
print('---------------- English Result ------------------')
CRF_check('English')
```

```
---------------- Chinese Result ------------------
              precision    recall  f1-score   support

   micro avg     0.9361    0.9439    0.9400      8437
   macro avg     0.7046    0.7315    0.7157      8437
weighted avg     0.9369    0.9439    0.9402      8437

---------------- English Result ------------------
              precision    recall  f1-score   support

   micro avg     0.8946    0.8562    0.8750      8603
   macro avg     0.8882    0.8324    0.8585      8603
weighted avg     0.8945    0.8562    0.8744      8603
```

Part 3: BiLSTM+CRF

BiLSTM层的输出为每一个标签的预测分值，然后将BiLSTM的输出值作为CRF层的输入，最后结果就是每个词的标签。虽然BiLSTM就可以完成标注工作，但是没有办法添加约束条件，通过CRF层就可以添加约束从而使预测的标签是合法的。通过CRF层前向计算损失函数值如下：

$$S_i = \sum_t (\sum_k^K \lambda_k f_k(y_t, X) + \sum_l^L \mu_l g_l(y_t, y_{t-1}, X))$$

$$loss = log(\sum_i e^{S_i}) - S_{real}$$

具体的代码实现见BiLSTM_CRF.py.

下面是运行结果：

```
---------------- Chinese Result ------------------
              precision    recall  f1-score   support

   micro avg     0.9466    0.9458    0.9462      8437
   macro avg     0.7240    0.7238    0.7230      8437
```

```
weighted avg     0.9467    0.9458    0.9462      8437

---------------- English Result ------------------

               precision    recall  f1-score   support


   micro avg     0.8861    0.8143    0.8487      8603
   macro avg     0.8493    0.8008    0.8224      8603
weighted avg     0.8889    0.8143    0.8489      8603
```

Part 3: 手写CRF

这里采用特征模版基于对训练数据的统计对特征函数进行构造。

具体代码实现详见my_CRF.py文件。