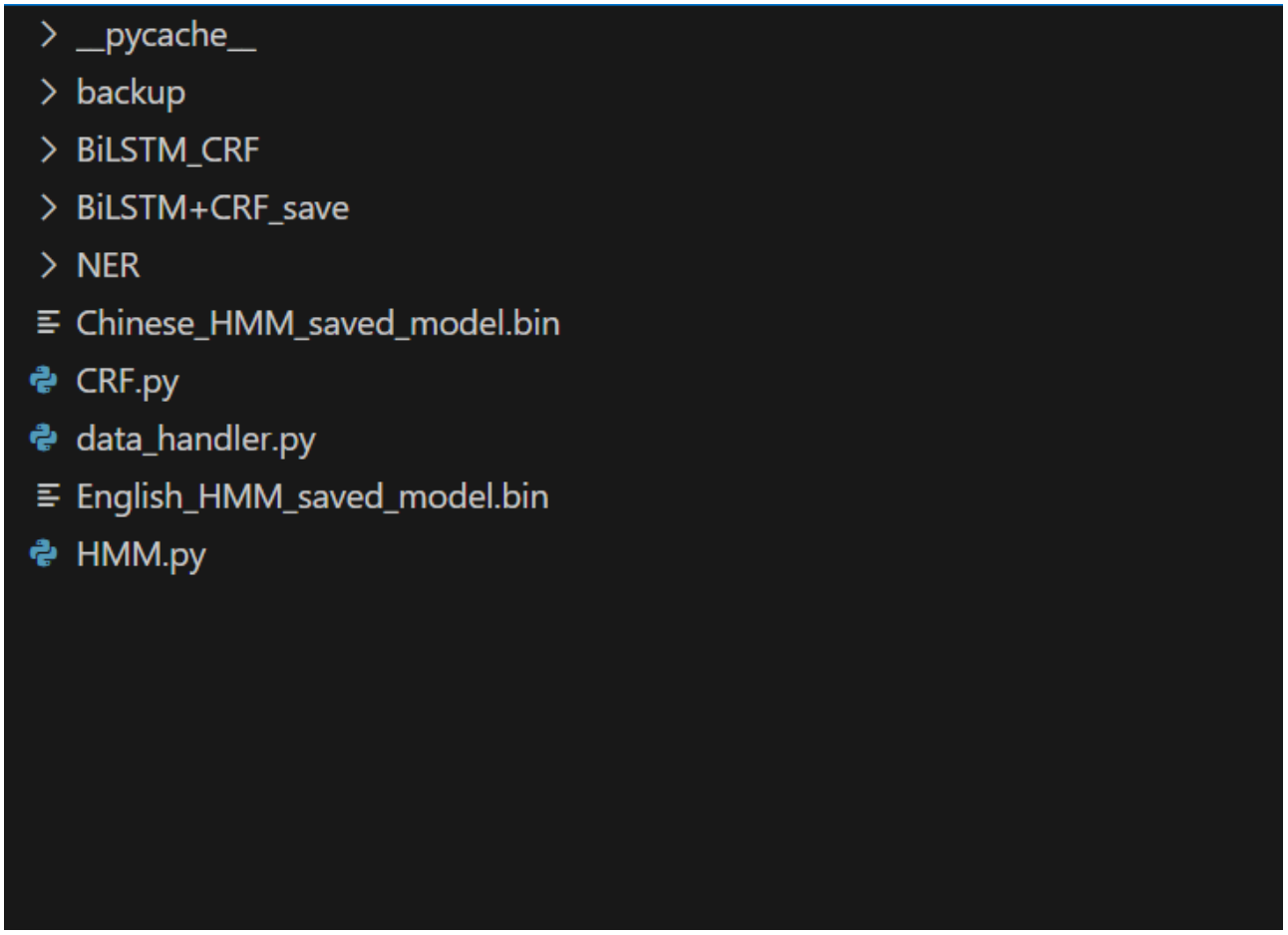
Lab2 命名实体识别 实验报告

22302010019　　陈星宇

一、文件结构

本次Lab相关代码均在Code文件夹下。Code中文件结构如下：

其中，HMM、CRF、BiLSTM_CRF/BiLSTM+CRF分别是3个部分的入口文件。HMM和BiLSTM的模型有另外保存，CRF未保存训练的模型。三者预测的输出结果都保存在NER文件夹中。backup文件夹中是某些已训练好的备份模型，可以忽略。



二、实现思路

1.HMM实现命名实体识别（NER）任务

（1）原始数据处理

原始数据通过data_handler的切割变化，转化为三重列表sentences，sentences型如:

`[[[word1,word2...],[lable1,lable2]],[[another word1,another word2....],[another lable1,another lable2...]]...]`

其中，word1,word2...构成了一句语料（此处，每句语料的切割是以空行为依据进行的）。word1对应的是标签lable1，word2对应的是标签lable2，以此类推。

（2）设计tags_dict字典

对于tags（中文时为33种，英文时为9种），建立与之一一对应的index，方便后续的矩阵操作。此处用字典存储，由8类（英文是4类）实体拓展为32类带有前缀的标签，再逐一加入tags_dict字典。

（3）三种矩阵的计算

三个矩阵初始化如下（所标注的矩阵大小以中文训练集为例）：

```
        self.transition_matrix = np.zeros((self.n_tag, self.n_tag))

        # 状态转移概率矩阵,33个标签相互转化, shape:(33, 33)

        self.emission_matrix  = np.zeros((self.n_tag, self.n_char))

        # 发射矩阵,共33个标签,每个标签有65535种可能的字符与之对应, shape:
(33, 65535)

        self.inital_matrix  = np.zeros(self.n_tag)
        # 初始矩阵,共33个标签,shape：(33,1)
```

值得一提的是，对于中文训练集，由于每个字是一个字符，故指定Unicode集大小（65535）作为发射矩阵的列数；对于英文训练集，在data_handler中指定了一个常量（大小约为常见的英文单词的数目）作为发射矩阵的列数。在HMM模型中有char2idx方法，对于前者，使用ord(char)将字符映射为唯一的Unicode码；对于后者，使用哈希函数映射到对应区间（但这种映射不是1对1的）。

三个矩阵在训练HMM时的更新如下：

```
def train(self, train_data):
        print('start trianing process')
        pre_tag = 'O'
        for i in range(len(train_data)):  # 用i遍历所有语料
            cur_sent=train_data[i][0] # 第i组语料的文字
            cur_tags=train_data[i][1] # 第i组语料的标签
            for j in range(len(cur_sent)):  # 用j遍历单组语料的所有字符
                cur_char = cur_sent[j]  # 取出当前字符
                cur_tag =  cur_tags[j]  # 取出当前标签
                self.transition_matrix[self.tags_dict[pre_tag]]
[self.tags_dict[cur_tag]] += 1  # 对transition_matrix矩阵中前一个标签-
>当前标签的位置加一
                pre_tag =cur_tag    # 更新前一个字符的标签
                self.emission_matrix [self.tags_dict[cur_tag]]
[self.char2idx(cur_char)] += 1  # 对发射矩阵中[标签][字符]的位置加一
                if j == 0:
                    # 对文本段的第一个字符统计初始矩阵
                    self.inital_matrix [self.tags_dict[cur_tag]] +=
1
                    continue
```

设置pre_tag,是为了记录上一个字的标签，对于第一个字，默认其上一个字的标签为'O'。更新的说明详见以上代码注释。

之后，还需对三个矩阵逐行对数归一化：

```
        # 防止数据下溢,对数据进行对数归一化

 self.transition_matrix=log_normalize(self.transition_matrix,False)
        self.emission_matrix =log_normalize(self.emission_matrix
,False)
        self.inital_matrix =log_normalize(self.inital_matrix ,True)
```

```
        '''
        省略部分代码
        '''
        def log_normalize(matrix,isOneDemension):
            matrix[matrix==0]=epsilon
            if(isOneDemension):
                return np.log(matrix) - np.log(np.sum(matrix))
            else:
                return np.log(matrix) - np.log(np.sum(matrix,
axis=1, keepdims=True))
```

在log_normalize函数中，还将矩阵中的所有零值替换为一个非常小的常数epsilon，这不仅是为了防止log（0）的错误，还有助于识别训练集中未出现的词（否则若有转移概率为0，那条路径在维特比解码时将永远不会被选择）。

（4）维特比解码

代码如下：

```
    def viterbi(self, s):
        num_chars = len(s) # 观测序列的长度
        delta = np.zeros((self.n_tag, num_chars)) # 一个矩阵，存储到达
每个时刻每个状态的最大概率
        paths = np.zeros((self.n_tag, num_chars), dtype=int) # 在每
个字符位置上最优路径的前驱标签

        # 初始化 delta 和路径
        first_char_idx = self.char2idx(s[0])
        # 初始化第一个字符的概率，等于初始概率加上发射概率
        delta[:, 0] = self.inital_matrix  + self.emission_matrix
[:, first_char_idx]


        # 动态规划填表
        for t in range(1, num_chars):
            char_idx = self.char2idx(s[t]) # 当前字符的索引
            max_prob = delta[:, t - 1][:, None] +
self.transition_matrix # 计算前一个时刻到当前时刻的所有状态转移的概率
            paths[:, t] = np.argmax(max_prob, axis=0) # 在每个字符位置
上最优路径的前驱标签
```

```
                delta[:, t] = np.max(max_prob, axis=0) +
self.emission_matrix [:, char_idx]

        # 回溯
        best_path = np.zeros(num_chars, dtype=int)
        best_path[-1] = np.argmax(delta[:, -1])
        # 反向循环，从序列的倒数第二个字符开始，一直到第一个字符结束
        for t in range(num_chars - 2, -1, -1):
            best_path[t] = paths[best_path[t + 1], t + 1]

        # 将索引转换为标签
        results = [self.idx2tag[idx] for idx in best_path]
        return results
```

先初始化第一个字符的概率，此后的字符以动态规划的方式填入paths中。其中，max_prob是一个矩阵，表示从前一个位置 t-1到当前位置 t 的所有可能转移的概率
np.argmax(max_prob, axis=0) 返回每列的最大值的索引，也即最优路径的前驱标签。而`delta[:, t] = np.max(max_prob, axis=0) + self.emission_matrix[:, char_idx]`更新了在位置 t上每个标签的最大概率。

然后可以进行回溯：`best_path[-1] = np.argmax(delta[:, -1])`确定了最后一位的标签，此后反向循环paths，从序列的倒数第二个字符开始，一直到第一个字符结束，构建best_path。

最后，把索引转换为标签，返回标签序列，即可完成预测。

（5）训练、验证和保存

HMM类中定义了save_model和load_model（静态）方法，用于保存和加载模型，但是目前的训练时间极短，故暂不使用这两个方法。完成预测后，再使用`print(metrics.classification_report(y_true,y_pred,labels=sorted_labels[1:], digits=4,zero_division=1))`打印详细分数报告,zero_division=1用于控制如何处理分母为零的情况（某标签未出现于测试集），即此时记录该标签预测准确率为100%（这不会影响micro F1-score的结果）

2.CRF实现命名实体识别（NER）任务

（1）原始数据处理

原始数据通过data_handler的切割变化，转化为三重列表sentences，sentences型如:

`[[[word1,word2...],[lable1,lable2]],[[another word1,another word2....],[another lable1,another lable2...]]...]`

其中，word1,word2...构成了一句语料（此处，每句语料的切割是以空行为依据进行的）。word1对应的是标签lable1，word2对应的是标签lable2，以此类推。

（2）文本特征提取

对一个字符（单词），我设计了如下的features：

```
    features = {
        'bias': 1.0,
        'word': word,
        'word.lower()': word.lower(),
        'word.isdigit()': word.isdigit(),
        'word.is_english()': is_english(word),
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'prefix-1': word[:1],
        'prefix-2': word[:2],
        'suffix-1': word[-1:],
        'suffix-2': word[-2:],
    }
```

其中，isupper()、word.istitle()、word.lower()是为了提高英文语料集的训练效果而设计的特征，对于中文语料集意义不大。设计窗口大小为5，相较于窗口大小为3时，micro F1-score有很大提升，原因应该是更大的窗口大小使得模型在特征提取时可以看到更多的上下文单词，从而提高模型对复杂语言现象的捕捉能力。

（3）CRF模型搭建

part2的CRF模型搭建使用了sklearn_crfsuite框架。其中，算法使用的是默认的lbfgs, c1、c2为正则化参数，c1增加稀疏性，c2惩罚权重的平方和，避免过拟合，经过测试，选定c1=0.4, c2=0.3。max_iterations为最大迭代次数，在100次后，正确率的提升已不明显。all_possible_transitions=True让模型考虑所有可能的转移，增加模型的鲁棒性；verbose=True控制训练过程中是否输出详细信息，有助于理解和调试模型。

```
crf_model = sklearn_crfsuite.CRF(c1=0.4, c2=0.3,
max_iterations=500,
                                 all_possible_transitions=True,
verbose=True)
```

（4）训练和验证

完成数据预处理后，使用sent2features和 sent2labels函数将整个句子转化为特征列表和标签列表，使用`crf_model.fit(X_train, y_train)`训练模型，`y_pred = crf_model.predict(X_dev)`进行预测。最后，同part1打印详细分数报告。

```
X_train = []
for sentence in train_sentences:
    features = sent2features(sentence[0])
    X_train.append(features)

y_train = []
for sentence in train_sentences:
    labels = sent2labels(sentence[1])
    y_train.append(labels)

X_dev = []
for sentence in valid_sentences:
    features = sent2features(sentence[0])
    X_dev.append(features)

y_dev = []
for sentence in valid_sentences:
    labels = sent2labels(sentence[1])
    y_dev.append(labels)
```

3.BiLSTM+CRF实现命名实体识别（NER）任务

（1）原始数据预处理

BiLSTM+CRF的数据预处理除了将原语料划分为sentences三重列表外，还完成了词表的建立：

```
def get_vocab():
    if os.path.exists(dictionary_path):
        with open(dictionary_path, 'rb') as fp:
            vocab = pickle.load(fp)
    else:
        sentences = load_and_split_data_from_file(train_file_path)
        vocab = {'PAD': 0, 'UNK': 1}
        for sentence in sentences:
            words = sentence[0]
            for word in words:
                if language == 'E':
                    if word not in vocab:
                        vocab[word] = len(vocab)
                elif language == 'C':
                    if word not in vocab and not is_english(word):
                        vocab[word] = len(vocab)
        with open(dictionary_path, 'wb') as fp:
            pickle.dump(vocab, fp)
    vocab_inv = {v: k for k, v in vocab.items()}
    return vocab, vocab_inv
```

这样的词表，解决了HMM模型中，对于未出现过的词处理不当的问题。（HMM模型中，只是在对数归一化阶段为其赋一个极小值，但也能达到精度要求）上面的函数生成或加载词表，并将其保存为pkl文件。词表的最大作用是帮助建立词到向量的映射。词表中有两个特殊项：`'PAD': 0, 'UNK': 1`PAD是用于填充batch未满的部分，UNK表示该词此前训练中未遇到过。

此外，data_processor中的CustomDataset 类是一个自定义的数据集类，它继承自`torch.utils.data.Dataset`，用于处理NER任务中的数据，并将其转换为适合模型训练的格式。它能够将数据中的每个单词转换为其在词汇表中的索引，如果单词不在词汇表中，则使用上述的 UNK代替。

CustomDataset 类的collect_batch方法将批次数据中的序列填充至相同长度，并转换为 PyTorch 张量。

（2）BiLSTM+CRF模型搭建

根据lab2要求，BiLSTM部分使用了pytorch框架，在BiLSTM_CRF的初始化函数中，主要定义了以下部分：

- **词嵌入层（Embedding Layer）**：将词汇表中的每个单词映射到一个固定维度的向量表示。（通过 nn.Embedding将输入的单词序列转换为固定维度的词向量表示）
- **双向LSTM层（BiLSTM Layer）**：通过两个方向的LSTM提取序列特征，并将其拼接。
- **线性层（Linear Layer）**：将BiLSTM的输出映射到标签空间，以生成每个时间步的发射分数，这些值将作为CRF的发射分数。
- **条件随机场（CRF Layer）**：用于序列标签的解码。

接下来，_get_lstm_features函数负责提取LSTM特征。它将输入句子转化为词嵌入，通过LSTM提取序列特征，最后通过线性层得到CRF的发射分数;forward函数根据模型的状态（训练、评估）决定是计算损失还是进行预测。在BiLSTM_CRF类中，还加入了dropout层，可以缓解过拟合。

另一方面，在CRF的初始化函数中，定义了转移矩阵self.transitions，并随机初始化了转移分数。forward_alg函数计算所有可能路径的总分数，_score_sentence函数计算给定标签序列的得分。总的来说，CRF层利用发射分数和转移分数进行序列解码，确保标签序列的一致性和全局最优。

发射矩阵和转移矩阵的更新过程如下：

发射矩阵表示在每个时间步上，每个标签的得分。这些得分通过LSTM的输出特征映射得到。在模型的前向传播过程中，LSTM的输出通过一个线性层 hidden2tag 转换为发射分数（emission scores）。这个线性层包含的权重参数即为发射矩阵的参数。这些参数在训练过程中会根据损失函数进行更新

损失函数 neg_log_likelihood计算了所有可能路径的总得分（通过前向算法 _forward_alg）与给定标签序列的得分（通过 _score_sentence）之间的差异。然后，利用反向传播更新模型的所有参数，包括发射矩阵和转移矩阵。在每个训练步骤中，通过计算损失函数来衡量模型的预测与实际标签之间的差异，然后通过反向传播算法计算损失函数相对于模型参数的梯度。使用优化器（Adam）更新模型的参数，包括发射矩阵和转移矩阵。

（3）训练和验证

先建立词表和标签-索引字典，再初始化CustomDataset，初始化BiLSTM+CRF模型（也可以选择从文件中加载已有模型），接着以dataloader的形式传入训练或测试函数中。训练使用Adam优化器：

```
optimizer = optim.Adam(model.parameters(), lr=learnning_rate,
weight_decay=weight_decay)
```

训练过程中，每个epoch进行一次测试，若得分更高则自动保存模型，否则不更新保存的模型，防止过拟合。

在评估过程中，使用 torch.no_grad() 禁用梯度计算，节省内存并加快计算速度。对验证数据集中的每一个批次中，将 words和 seq_len移动到指定设备，接着获取模型对当前批次的预测标签。最后将真实标签和预测标签转换为可读的标签名称，并添加到 all_label和 all_pred 列表中（后面还需将嵌套列表展开成一个一维列表，以便计算评估指标）。

（4）对加快训练速度的尝试

这一部分参考了网络上的实现方法。设置device为"cuda 0"，把张量转移到 GPU 进行计算，因为 GPU 通常比 CPU 更适合进行大量并行计算，从而显著提高了计算速度，在其他条件相同的前提下，对于中文训练集，一个batch的训练时间由0.76秒下降到了0.14秒左右，有了显著的提升。

三、训练结果

以下的测试，若无特别说明，训练集和验证集相同。测试结果来自check.py的运行结果，或除了添加了zero_division=1外，与之参数完全一致的训练信息的打印结果。

1.HMM

中文：

模型已保存到文件

	precision	recall	f1-score	support
B-NAME	0.9524	0.9988	0.9751	861
M-NAME	0.9331	0.9797	0.9558	740
E-NAME	0.9369	0.9826	0.9592	861
S-NAME	0.9296	0.7253	0.8148	91
B-CONT	0.9886	1.0000	0.9943	260
M-CONT	0.9940	1.0000	0.9970	499
E-CONT	0.9886	1.0000	0.9943	260
S-CONT	1.0000	1.0000	1.0000	0
B-EDU	0.9241	0.9790	0.9508	858
M-EDU	0.9611	0.9818	0.9713	1536
E-EDU	0.9351	0.9907	0.9621	858
S-EDU	1.0000	1.0000	1.0000	0
B-TITLE	0.8759	0.9012	0.8884	6296
M-TITLE	0.8900	0.9187	0.9041	14813
E-TITLE	0.9546	0.9819	0.9681	6296
S-TITLE	1.0000	1.0000	1.0000	0
B-ORG	0.9099	0.9587	0.9337	4603
M-ORG	0.9469	0.9470	0.9470	33762
E-ORG	0.8487	0.8922	0.8699	4603
S-ORG	1.0000	1.0000	1.0000	1
B-RACE	1.0000	1.0000	1.0000	112
M-RACE	1.0000	0.8333	0.9091	6
E-RACE	0.9911	0.9911	0.9911	112
S-RACE	1.0000	1.0000	1.0000	3
B-PRO	0.7222	0.9512	0.8211	287
M-PRO	0.7430	0.9505	0.8340	666
E-PRO	0.7354	0.9686	0.8361	287
S-PRO	1.0000	1.0000	1.0000	0
B-LOC	0.7419	0.9787	0.8440	47
M-LOC	0.8171	1.0000	0.8994	143
E-LOC	0.7581	1.0000	0.8624	47
S-LOC	1.0000	1.0000	1.0000	0
micro avg	0.9186	0.9419	0.9301	78908
macro avg	0.9212	0.9660	0.9401	78908
weighted avg	0.9199	0.9419	0.9305	78908

英文：

	precision	recall	f1-score	support
B-PER	0.9730	0.9595	0.9662	6600
I-PER	0.9845	0.9932	0.9888	4528
B-ORG	0.8959	0.9337	0.9144	6321
I-ORG	0.9423	0.9168	0.9294	3704
B-LOC	0.9305	0.9462	0.9383	7140
I-LOC	0.9155	0.9274	0.9214	1157
B-MISC	0.9611	0.9421	0.9515	3438
I-MISC	0.9035	0.9489	0.9257	1155
micro avg	0.9419	0.9486	0.9452	34043
macro avg	0.9383	0.9460	0.9420	34043
weighted avg	0.9424	0.9486	0.9453	34043

2.CRF

中文（迭代**50**次）：

micro F1 score: 0.9919446328184911

	precision	recall	f1-score	support
B-CONT	1.0000	0.9962	0.9981	260
E-CONT	1.0000	0.9962	0.9981	260
M-CONT	1.0000	0.9980	0.9990	499
B-EDU	0.9977	0.9942	0.9959	858
E-EDU	0.9965	0.9930	0.9947	858
M-EDU	0.9967	0.9974	0.9971	1536
B-LOC	1.0000	0.9787	0.9892	47
E-LOC	1.0000	0.9787	0.9892	47
M-LOC	1.0000	0.9650	0.9822	143
B-NAME	0.9965	0.9965	0.9965	861
E-NAME	0.9954	0.9954	0.9954	861
M-NAME	0.9973	0.9824	0.9898	740
S-NAME	0.9785	1.0000	0.9891	91
B-ORG	0.9941	0.9900	0.9921	4603
E-ORG	0.9852	0.9809	0.9830	4603
M-ORG	0.9934	0.9959	0.9946	33762
S-ORG	1.0000	0.0000	0.0000	1
B-PRO	0.9861	0.9861	0.9861	287
E-PRO	0.9861	0.9861	0.9861	287
M-PRO	0.9864	0.9835	0.9850	666
B-RACE	1.0000	1.0000	1.0000	112
E-RACE	1.0000	1.0000	1.0000	112
M-RACE	1.0000	1.0000	1.0000	6
S-RACE	1.0000	0.6667	0.8000	3
B-TITLE	0.9871	0.9855	0.9863	6296
E-TITLE	0.9970	0.9954	0.9962	6296
M-TITLE	0.9890	0.9869	0.9880	14813
micro avg	0.9922	0.9917	0.9919	78908
macro avg	0.9949	0.9418	0.9486	78908
weighted avg	0.9922	0.9917	0.9919	78908

英文（迭代**50**次）：

```
micro F1 score: 0.9967872097449563
              precision    recall  f1-score   support

       B-LOC     0.9889    0.9873    0.9881      7140
       I-LOC     0.9852    0.9775    0.9813      1157
      B-MISC     0.9895    0.9607    0.9749      3438
      I-MISC     0.9697    0.9714    0.9706      1155
       B-ORG     0.9849    0.9780    0.9814      6321
       I-ORG     0.9756    0.9835    0.9796      3704
       B-PER     0.9876    0.9864    0.9870      6600
       I-PER     0.9903    0.9971    0.9937      4528

   micro avg     0.9859    0.9827    0.9843     34043
   macro avg     0.9840    0.9802    0.9821     34043
weighted avg     0.9859    0.9827    0.9843     34043
```

3.BiLTSM+CRF

中文（迭代**5**次，其他学习参数详见提交代码）：

```
Test done, micro F1 score: 0.9815957130474945
              precision    recall  f1-score   support

      B-NAME     0.9806    0.9965    0.9885       861
      M-NAME     0.9635    0.9986    0.9808       740
      E-NAME     0.9862    0.9942    0.9902       861
      S-NAME     0.9891    1.0000    0.9945        91
      B-CONT     1.0000    0.9962    0.9981       260
      M-CONT     0.9940    1.0000    0.9970       499
      E-CONT     0.9962    1.0000    0.9981       260
      S-CONT     1.0000    1.0000    1.0000         0
       B-EDU     0.9884    0.9907    0.9895       858
       M-EDU     0.9902    0.9902    0.9902      1536
       E-EDU     0.9871    0.9848    0.9860       858
       S-EDU     1.0000    1.0000    1.0000         0
     B-TITLE     0.9723    0.9709    0.9716      6296
     M-TITLE     0.9888    0.9573    0.9728     14813
     E-TITLE     0.9910    0.9943    0.9926      6296
     S-TITLE     1.0000    1.0000    1.0000         0
       B-ORG     0.9669    0.9900    0.9783      4603
       M-ORG     0.9795    0.9945    0.9870     33762
       E-ORG     0.9587    0.9676    0.9631      4603
       S-ORG     0.5000    1.0000    0.6667         1
      B-RACE     0.9912    1.0000    0.9956       112
      M-RACE     1.0000    1.0000    1.0000         6
      E-RACE     1.0000    1.0000    1.0000       112
      S-RACE     1.0000    1.0000    1.0000         3
       B-PRO     0.9782    0.9373    0.9573       287
       M-PRO     0.9757    0.9640    0.9698       666
       E-PRO     0.9758    0.9826    0.9792       287
       S-PRO     1.0000    1.0000    1.0000         0
       B-LOC     0.9565    0.9362    0.9462        47
       M-LOC     0.9929    0.9790    0.9859       143
       E-LOC     0.9375    0.9574    0.9474        47
       S-LOC     1.0000    1.0000    1.0000         0

   micro avg     0.9801    0.9831    0.9816     78908
   macro avg     0.9700    0.9870    0.9758     78908
weighted avg     0.9802    0.9831    0.9816     78908
```

英文（迭代5次，其他学习参数详见提交代码）：

```
Test done, micro F1 score: 0.9752668882157456
              precision    recall  f1-score   support

       B-PER      0.984     0.986     0.985      6600
       I-PER      0.992     0.992     0.992      4528
       B-ORG      0.968     0.970     0.969      6321
       I-ORG      0.953     0.964     0.958      3704
       B-LOC      0.983     0.985     0.984      7140
       I-LOC      0.962     0.940     0.951      1157
      B-MISC      0.981     0.954     0.968      3438
      I-MISC      0.956     0.913     0.934      1155

   micro avg      0.976     0.974     0.975     34043
   macro avg      0.972     0.963     0.968     34043
weighted avg      0.976     0.974     0.975     34043
```