pj2 part2 CRF模型

实验原理

1. Features to Weights：CRF与HMM的一大不同是可以定义更多的模板，在template.utf8中给出了很多模板，unigram和bigram分别表示当前位置单词的标签 li 以及 li 和前一个单词的标签 li−1的关系。

接下来，我们要为每一个feature function fk 赋予一个权重 λk。给一个句子 s ， s可以对应许许多多的标签序列 l 。因此，我们可以通过将所有words的weighted features相加，对每一种标签序列l打分：

$$score(l|s) = \sum_{k=1}^{m} \sum_{i=1}^{n} \lambda_k f_k(s, i, l_i, l_{i-1})$$

上式中其中，句子s长度为n，feature functions的数量为m。

最后，我们将这些scores转换为概率 P(l|s) ，就会得到下式，也就是CRF的核心表达式：

$$P(l|s) = \frac{exp[score(l|s)]}{\sum_{l'} exp[score(l'|s)]} = \frac{exp[\sum_{k=1}^{m} \sum_{i=1}^{n} \lambda_k f_k(s,i,l_i,l_{i-1})]}{\sum_{l'} exp[\sum_{k=1}^{m} \sum_{i=1}^{n} \lambda_k f_k(s,i,l'_i,l'_{i-1})]}.$$

2. CRF的目标函数：

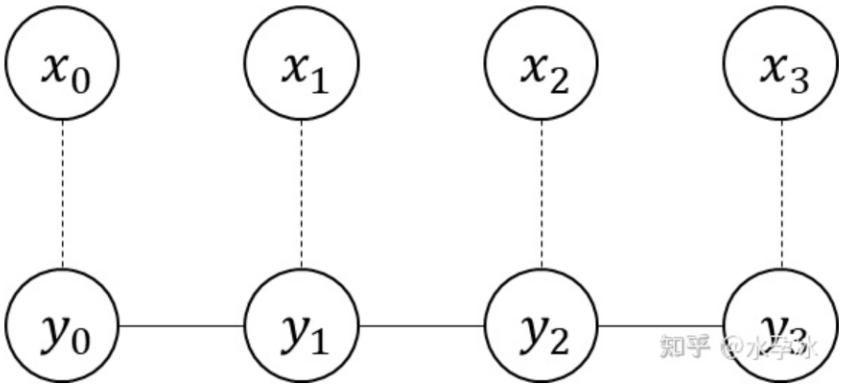

图1 观测序列x和隐状态序列y之间的关系

由于我们的最终目标是找到能使得score值最大的隐状态序列，因此我们需要计算每个隐状态序列 y 对应的 P(y|x)。如果我们已经自定义好了一组feature functions，我们唯一不知道的就是每个feature functions fk 的权重 wk 。我们就要确定CRF的目标函数（也就是损失函数），通过优化目标函数，达到确定 wk 的目的。

$$\max_{w \in R^+} \prod_{x,y} P_w(y|x)^{\widetilde{P}(x,y)},$$

上式即为CRF的优化目标函数，其中P~(x,y) 是 x 和 y 的真实联合分布。

3. 利用最大似然的方法得到最终的损失函数：

$$\min_{w \in R^+} f(w) = -\sum_{x,y} \widetilde{P}(x,y) \log P_w(y|x),$$

4. 具体的计算方法使用了L-BFGS算法：

1. 选定初始权重向量 w^0 ，取 $G_0 = I$ ，置 $k = 0$ （表示第 k 次迭代）。
2. 计算 $g^k = g(w^k)$ (g^k 是每一次迭代时，我们输入L-BFGS的值)。若 $g^k = 0$ ，则计算结束，转7；否则转3。
3. 进行一维搜索，求 λ^k ，使得（这里的 λ^k 是避免牛顿法存在的不收敛情况，而加入的步长因子） $f(w^k - \lambda^k \cdot \frac{g^k}{B^k}) = \min_{\lambda \geq 0} f(w^k - \lambda \cdot \frac{g^k}{B^k})$。
4. 进行 w 更新： $w^{k+1} = w^k - \lambda \cdot \frac{g^k}{B^k}$ 。
5. 计算 $g^{k+1} = g(w^{k+1})$ （再一次调用我们的代码计算梯度）。若 $g^{k+1} = 0$ ，则计算结束，转7；否则，更新 B^{k+1} ，
   $B^{k+1} = B^k + \frac{(g^{k+1}-g^k)\cdot(g^{k+1}-g^k)^T}{(g^{k+1}-g^k)\cdot(w^{k+1}-w^k)^T} - \frac{B^k\cdot(w^{k+1}-w^k)\cdot(w^{k+1}-w^k)^T\cdot B_k}{(w_{k+1}-w^k)^T\cdot B_k\cdot(w^{k+1}-w^k)}$ 。
6. 置 $k = k + 1$ ，转3。
7. 输出此时的 w 。

总结：我们在调用L-BFGS包时，需要输入提前初始化好的权重向量 w^0 ，然后在每一轮迭代时我们需要不断计算好 $g(w)$ 和 $f(w)$ 。

Code

获取标签字典、词汇字典、数据集的方式与HMM中相同，在此不再赘述。

feature　　Plain Text | 复制代码

```
def word2features(sent, i):
    word = sent[i]
    prev_word = '<s>' if i == 0 else sent[i-1]
    next_word = '</s>' if i == (len(sent)-1) else sent[i+1]
    features = {
        'w': word,
        'w-1': prev_word,
        'w+1': next_word,
        'w-1:w': prev_word+word,
        'w:w+1': word+next_word,
        'bias': 1
    }
    return features


def sent2features(sent):
    return [word2features(sent, i) for i in range(len(sent))]
```

这里定义了特征模板。对于给出的template.utf8来说有一定的简化，unigram上只考虑了本位置及前后位置上的文本，bigram上只考虑了当前位置文本与前后相邻文本的关系。这里要注意，句子的首位要单独定义prev和next，分别定义成sentence的其实和结束标签<s>和</s>.

CRFModel Plain Text 复制代码

```
class CRFModel(object):
    def __init__(self, algorithm='lbfgs', c1=0.1, c2=0.1,
                 max_iterations=100, all_possible_transitions=False):
        self.crf = CRF(algorithm=algorithm,
                       c1=c1,
                       c2=c2,
                       max_iterations=max_iterations,
                       all_possible_transitions=all_possible_transition
s)

    def train(self, train_words, train_tags):
        features = [sent2features(s) for s in train_words]
        self.crf.fit(features, train_tags)

    def val(self, val_words, word_dict, tag_dict, out_path):
        f = open(out_path, "w", encoding="utf-8")
        features = [sent2features(s) for s in val_words]
        preds = self.crf.predict(features)
        for i, words in enumerate(val_words):
            for j in range(len(words)): # find the key
                f.write(words[j] + " " + preds[i][j] + "\n")
            if i!=len(val_words)-1:
                f.write("\n")
        f.close()
```

上述代码是CRF的类定义，利用了crfsuite库。

- 使用CRF算法来训练一个条件随机场模型，其中参数algorithm用来指定训练时所使用的优化算法，c1和c2用来控制L1和L2正则化的强度，max_iterations指定模型的最大迭代次数，all_possible_transitions用来控制是否考虑所有可能的转移。
- 将训练数据转换成特征向量的形式，通过调用sent2features函数将每个句子转换成对应的特征矩阵。
- 使用上一步得到的特征向量和标记序列train_tags来训练CRF模型。CRF.fit方法会自动调用所选的优化算法来优化模型的权重参数，以使模型在训练数据上的损失函数最小化。

实验结果

中文validation上的精度：

	precision	recall	f1-score	support
B-NAME	0.9901	0.9804	0.9852	102
M-NAME	1.0000	0.9733	0.9865	75
E-NAME	0.9901	0.9804	0.9852	102
S-NAME	1.0000	1.0000	1.0000	8
B-CONT	1.0000	1.0000	1.0000	33
M-CONT	1.0000	1.0000	1.0000	64
E-CONT	1.0000	1.0000	1.0000	33
S-CONT	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	0.0000	0.0000	0.0000	1
B-PRO	0.8095	0.9444	0.8718	18
M-PRO	0.7021	1.0000	0.8250	33
E-PRO	0.8571	1.0000	0.9231	18
B-LOC	1.0000	1.0000	1.0000	2
M-LOC	1.0000	1.0000	1.0000	6
E-LOC	1.0000	1.0000	1.0000	2
S-LOC	0.0000	0.0000	0.0000	0
micro avg	0.9370	0.9519	0.9444	8437
macro avg	0.7170	0.7330	0.7238	8437
weighted avg	0.9376	0.9519	0.9445	8437

英文validation上的精度：

```
PS D:\大学学习\大三\大三下\人工智能\Project2\NER> & C:/Users/16367/AppData/Local/Programs/Python/Python311/python.exe d:/大学学习/大三/大三下/人工智能/Project2/NER/check.py
              precision    recall  f1-score   support

       B-PER     0.9414    0.7155    0.8131      1842
       I-PER     0.9321    0.8187    0.8717      1307
       B-ORG     0.9313    0.6667    0.7771      1341
       I-ORG     0.9206    0.7257    0.8116       751
       B-LOC     0.9566    0.7556    0.8443      1837
       I-LOC     0.9242    0.7588    0.8333       257
      B-MISC     0.9598    0.7245    0.8257       922
      I-MISC     0.9729    0.6214    0.7584       346

   micro avg     0.9422    0.7315    0.8236      8603
   macro avg     0.9423    0.7233    0.8169      8603
weighted avg     0.9426    0.7315    0.8227      8603
```

实验思考

在CRF目标函数的表达式中，P~(x,y) 是 x 和 y 的真实联合分布，而不使用真实条件分布 P~(y|x)，这是因为P~(y|x)由于没有考虑到 P~(x) 分布的影响。

在一般情况下，我们利用大量数据训练CRF模型时，所有的观测序列肯定都是不同的，且每一个观测序列都肯定对应一个真实隐状态序列，则所有的观测序列出现的概率都相同，即 P~(x)=1/N；一旦训练数据中存在大量的重复观测序列如x'，则说明观测序列 x' 在实际中出现的概率更高，那么CRF模型会更倾向于优化 Pw(y|x')P~(y|x') 这一项，这也更符合实际。说明了训练数据的分布要尽可能接近真实数据的分布，CRF模型才会训练的越好。

参考链接

https://zhuanlan.zhihu.com/p/483820319?utm_id=0