

NLP.TM | 命名实体识别基线 BiLSTM+CRF (下)

原创 机智的叉烧 CS的陋室 2019-07-12



点击上方蓝色文字立刻订阅精彩

我愿意平凡的陪在你身旁

王七七 - 我愿意平凡的陪在你身旁



【NLP.TM

】

本人有关自然语言处理和文本挖掘方面的学习和笔记，欢迎大家关注。

往期回顾：

- [NLP.TM | tensorflow做基础的文本分类](#)
- [NLP.TM | Keras做基本的文本分类](#)
- [NLP.TM | 再看word2vector](#)
- [NLP.TM | GloVe模型及其Python实现](#)
- [NLP.TM | 我的NLP学习之路](#)

命名实体识别是继文本分类之后的一个重要任务。在语言学方面，分词、词性标注、句法分析等，在工业应用方面，则有实体抽取等，其实都用到了命名实体识别技术，本文将介绍命名实体识别任务以及其重要的基线模型BiLSTM+CRF。

另外由于文章太长，所以我分为两块，理论和思路我放一篇，实现我放另一篇，本文是下篇，开始讲实现啦，上篇在这里：

NLP.TM | 命名实体识别基线 BiLSTM+CRF (上)

懒人目录

- 数据预处理
- 模型部分
- 训练主程序

■ 存在缺陷

开始说实现吧，这里我是踩了太多太多的坑，代码自己实现了一遍，我之所以一周没发文(咳咳恩)就是因为这里的暗坑实在太多了(可能也是我水平不够吧，希望大家看着代码能指导一下)，那么下面就来讨论一下怎么实现吧。

开始之前，感谢这位优秀的同志在github上开源的代码，模型一块的代码都是参照他的，另外用的也是他提供的人民日报数据。

<https://github.com/buppt/ChineseNER>

完整代码见我的github：

<https://gitee.com/chashaozgr/noteLibrary/tree/master/nlptrial/ner/src/bilstmcrf>

数据预处理

要做分类问题，当然要整理出X和Y的基本形式，而对于原始数据是这样的：

```
19980101-01-001-001/m 迈向/v 充满/v 希望/n 的/u 新/a 世纪/n 一/w 一九九八年/t 新年/t 讲话/n (/w 附/v
图片/n 1/m 张/q ) /w
19980101-01-001-002/m 中共中央/nt 总书记/n 、 /w 国家/n 主席/n 江/nr 泽民/nr
19980101-01-001-003/m (/w 一九九七年/t 十二月/t 三十一日/t ) /w
19980101-01-001-004/m 1 2月/t 3 1日/t , /w 中共中央/nt 总书记/n 、 /w 国家/n 主席/n 江/nr 泽民/nr 发表/
v 1 9 9 8年/t 新年/t 讲话/n 《/w 迈向/v 充满/v 希望/n 的/u 新/a 世纪/n 》/w 。 /w (/w 新华社/nt 记者/
n 兰/nr 红光/nr 摄/Vg ) /w
19980101-01-001-005/m 同胞/n 们/k 、 /w 朋友/n 们/k 、 /w 女士/n 们/k 、 /w 先生/n 们/k : /w
19980101-01-001-006/m 在/p 1 9 9 8年/t 来临/v 之际/f , /w 我/r 十分/m 高兴/a 地/u 通过/p [中央/n 人民/
n 广播/vn 电台/n]nt 、 /w [中国/ns 国际/n 广播/vn 电台/n]nt 和/c [中央/n 电视台/n]nt , /w 向/p 全国/n
各族/r 人民/n , /w 向/p [香港/ns 特别/a 行政区/n]ns 同胞/n 、 /w 澳门/ns 和/c 台湾/ns 同胞/n 、 /w 海外/
s 侨胞/n , /w 向/p 世界/n 各国/r 的/u 朋友/n 们/k , /w 致以/v 诚挚/a 的/u 问候/vn 和/c 良好/a 的/u
祝愿/vn ! /w
19980101-01-001-007/m 1 9 9 7年/t , /w 是/v 中国/ns 发展/vn 历史/n 上/f 非常/d 重要/a 的/u 很/d 不/d
平凡/a 的/u 一/m 年/q 。 /w 中国/ns 人民/n 决心/d 继承/v 邓/nr 小平/nr 同志/n 的/u 遗志/n , /w 继续/v
把/p 建设/v 有/v 中国/ns 特色/n 社会主义/n 事业/n 推向/v 前进/v 。 /w [中国/ns 政府/n]nt 顺利/ad 恢复/v
对/p 香港/ns 行使/v 主权/n , /w 并/c 按照/p “/w 一国两制/j ”/w 、 /w “/w 港人治港/l ”/w 、 /w 高度/d
自治/v 的/u 方针/n 保持/v 香港/ns 的/u 繁荣/an 稳定/an 。 /w [中国/ns 共产党/n]nt 成功/a 地/u 召开/v 了/
```

因此需要有比较复杂的工作才能把数据进行转化，注意，对于数据一定一定要认真探索，这样才能在清晰目标的情况下准确无误地把杂乱的数据转化为你的目标形式，同时需要注意避免引入新的混乱因素。

上代码，此处代码借鉴了上面提到的github作者的内容。

```
# 初步提取信息
fout = open(SOURCE_2_DATA, "w")
with open(SOUTCE_DATA, "r") as f:
    for line in f:
        line = line.split(' ')
        i = 1
        while i < len(line) - 1:
```

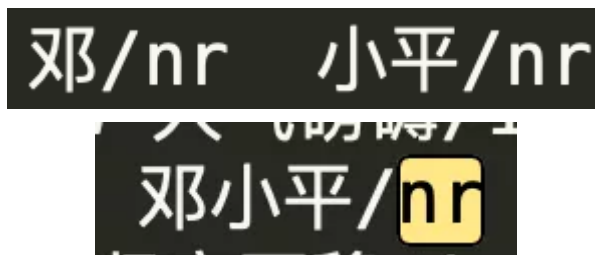
```

if line[i][0] == '[':
    fout.write(line[i].split('/')[0][1:])
    i += 1
    while i < len(line) - 1 and line[i].find(']') == -1:
        if line[i] != '':
            fout.write(line[i].split('/')[0])
            i += 1
        fout.write(line[i].split('/')[0].strip() + '/' +
                    line[i].split('/')[1][-2:] + ' ')
    elif line[i].split('/')[1] == 'nr':
        word = line[i].split('/')[0]
        i += 1
        if i < len(line) - 1 and line[i].split('/')[1] == 'nr':
            fout.write(word + line[i].split('/')[0] + '/nr ')
        else:
            fout.write(word + '/nr ')
            continue
    else:
        fout.write(line[i] + ' ')
    i += 1
fout.write('\n')
fout.close()

```

首先是，初步提取文本信息。

- 剔除"19980101-01-001-001/m"之类的有关时间、行数之类的信息
- 方括号处理
- 有关nr的切词，此处可以看到nr的切词其实对姓名是分开的，但实际上我们要把他们组合起来。



```

# 只保留nr、ns和nt
fout = open(SOURCE_3_DATA, "w")
with open(SOURCE_2_DATA, "r") as f:
    for line in f:
        line = line.split(' ')
        i = 0
        while i < len(line) - 1:
            if line[i] == '':
                i += 1
                continue
            word = line[i].split('/')[0]
            tag = line[i].split('/')[1]
            if tag == 'nr' or tag == 'ns' or tag == 'nt':

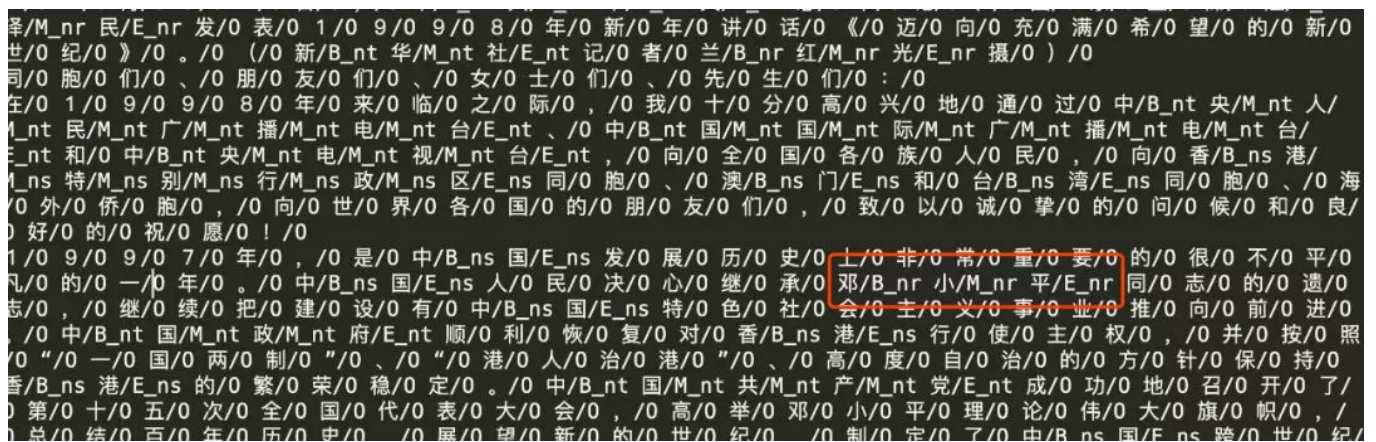
```

```

fout.write(word[0]+"/B_"+tag+" ")
for j in word[1:len(word)-1]:
    if j!=' ':
        fout.write(j+"/M_"+tag+" ")
    fout.write(word[-1]+"/E_"+tag+" ")
else:
    for wor in word:
        fout.write(wor+' /0 ')
    i+=1
    fout.write('\n')
fout.close()

```

然后，根据实际问题，将标签转化，这里只保留nr、ns和nt，另外我们把简单的标签再细分为B、M、E三种格式，并将词汇转化为单字的形式。



译/M_nr 民/E_nr 发/0 表/0 1/0 9/0 9/0 8/0 年/0 新/0 年/0 讲/0 话/0 《/0 迈/0 向/0 充/0 满/0 希/0 望/0 的/0 新/0
 世/0 纪/0 》/0 。/0 (/0 新/B_nt 华/M_nt 社/E_nt 记/0 者/0 兰/B_nr 红/M_nr 光/E_nr 摄/0) /0
 同/0 胞/0 们/0 、/0 朋/0 友/0 们/0 、/0 女/0 士/0 们/0 、/0 先/0 生/0 们/0 : /0
 在/0 1/0 9/0 9/0 8/0 年/0 来/0 临/0 之/0 际/0 , /0 我/0 十/0 分/0 高/0 兴/0 地/0 通/0 过/0 中/B_nt 央/M_nt 人/
 M_nt 民/M_nt 广/M_nt 播/M_nt 电/M_nt 台/E_nt 、/0 中/B_nt 国/M_nt 国/M_nt 际/M_nt 广/M_nt 播/M_nt 电/M_nt 台/
 E_nt 和/0 中/B_nt 央/M_nt 电/M_nt 视/M_nt 台/E_nt , /0 向/0 全/0 国/0 各/0 族/0 人/0 民/0 , /0 向/0 香/B_ns 港/
 M_ns 特/M_ns 别/M_ns 行/M_ns 政/M_ns 区/E_ns 同/0 胞/0 、/0 澳/B_ns 门/E_ns 和/0 台/B_ns 湾/E_ns 同/0 胞/0 、/0 海
 外/0 侨/0 胞/0 , /0 向/0 世/0 界/0 各/0 国/0 的/0 朋/0 友/0 们/0 , /0 致/0 以/0 诚/0 挚/0 的/0 问/0 候/0 和/0 良/
 好/0 的/0 祝/0 愿/0 ! /0
 1/0 9/0 9/0 7/0 年/0 , /0 是/0 中/B_ns 国/E_ns 发/0 展/0 历/0 史/0 上/0 非/0 常/0 重/0 要/0 的/0 很/0 不/0 平/0
 凡/0 的/0 一/0 年/0 。/0 中/B_ns 国/E_ns 人/0 民/0 决/0 心/0 继/0 承/0 邓/B_nr 小/M_nr 平/E_nr 同/0 志/0 的/0 遗/0
 志/0 , /0 继/0 续/0 把/0 建/0 设/0 有/0 中/B_ns 国/E_ns 特/0 色/0 社/0 会/0 主/0 义/0 事/0 业/0 推/0 向/0 前/0 进/0
 , /0 中/B_nt 国/M_nt 政/M_nt 府/E_nt 顺/0 利/0 恢/0 复/0 对/0 香/B_ns 港/E_ns 行/0 使/0 主/0 权/0 , /0 并/0 按/0 照
 “/0 一/0 国/0 两/0 制/0 ”/0 、/0 “/0 港/0 人/0 治/0 港/0 ”/0 、/0 高/0 度/0 自/0 治/0 的/0 方/0 针/0 保/0 持/0
 香/B_ns 港/E_ns 的/0 繁/0 荣/0 稳/0 定/0 。/0 中/B_nt 国/M_nt 共/M_nt 产/M_nt 党/E_nt 成/0 功/0 地/0 召/0 开/0 了/
 第/0 十/0 五/0 次/0 全/0 国/0 代/0 表/0 大/0 会/0 , /0 高/0 举/0 邓/0 小/0 平/0 理/0 论/0 伟/0 大/0 旗/0 帜/0 , /0
 总/0 结/0 百/0 年/0 历/0 史/0 , /0 展/0 望/0 新/0 的/0 世/0 纪/0 , /0 制/0 定/0 了/0 中/B_ns 国/E_ns 跨/0 世/0 纪/0

```

# 删除标点符号，断句
fout = open(SOURCE_4_DATA, "w")
with open(SOURCE_3_DATA, "r") as f:
    texts = f.read()
    sentences = re.split('[,。！？、‘’“”:]|[0]', texts)
    for sentence in sentences:
        if sentence != " ":
            fout.write(sentence.strip()+'\n')
fout.close()

```

删除标点符号是一个比较常规的操作，至于断句，此处有针对模型本身的思考。

- 机器学习训练非常要求数据量，而这个数据量并不是体现在存储大小上，而是数据的条数，即样本量上，长文本虽然更有利于分析，但是对于机器学习模型而言，样本量更为重要，因此通过断句的方式，将一条样本转化为多条样本在当前条件下更为合适。

断句前有23064个样本，断句后有154949个样本，数据量的数据集直接上升一个数量级。

```

# 数据集最终构建
datas = []
labels = []
linedata=[]

```

```

linelabel=[]
tags = {}
tags[''] = 0
tag_id_tmp = 1
words = {}
words["unk_"] = 0
word_id_tmp = 1
f = open(SOURCE_4_DATA, "r")
for line in f:
    line = line.split()
    linedata=[]
    linelabel=[]
    numNotO=0
    for word in line:
        word = word.split('/')
        linedata.append(word[0])
        linelabel.append(word[1])
        if word[0] not in words:
            words[word[0]] = word_id_tmp
            word_id_tmp = word_id_tmp + 1
        if word[1] not in tags:
            tags[word[1]] = tag_id_tmp
            tag_id_tmp = tag_id_tmp + 1
        if word[1]!='0':
            numNotO+=1
    if numNotO!=0:
        datas.append(linedata)
        labels.append(linelabel)
words[""] = word_id_tmp
f.close()

# word&id
fout_w2id = open("../data/people_daily/word2id_dict", "w")
fout_id2w = open("../data/people_daily/id2word_dict", "w")
for word_key in words.keys():
    fout_w2id.write("%s\t%s\n" % (word_key, words[word_key]))
    fout_id2w.write("%s\t%s\n" % (words[word_key], word_key))
fout_w2id.close()
fout_id2w.close()

# tag&id
fout_t2id = open("../data/people_daily/tag2id_dict", "w")
fout_id2t = open("../data/people_daily/id2tag_dict", "w")
for tag_key in tags.keys():
    fout_t2id.write("%s\t%s\n" % (tag_key, tags[tag_key]))
    fout_id2t.write("%s\t%s\n" % (tags[tag_key], tag_key))
fout_t2id.close()
fout_id2t.close()

x_train,x_test, y_train, y_test = train_test_split(datas, labels, test_size=0.2, random_s
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.25,

with open("../data/people_daily/x_train", "w") as f:

```

```
for idx in range(len(x_train)):
    write_str = "%s\n" % ("\t".join([str(i) for i in x_train[idx]]))
    f.write(write_str)
with open("../data/people_daily/x_test", "w") as f:
    for idx in range(len(x_test)):
        write_str = "%s\n" % ("\t".join([str(i) for i in x_test[idx]]))
        f.write(write_str)
with open("../data/people_daily/x_valid", "w") as f:
    for idx in range(len(x_valid)):
        write_str = "%s\n" % ("\t".join([str(i) for i in x_valid[idx]]))
        f.write(write_str)
with open("../data/people_daily/y_train", "w") as f:
    for idx in range(len(y_train)):
        write_str = "%s\n" % ("\t".join([str(i) for i in y_train[idx]]))
        f.write(write_str)
with open("../data/people_daily/y_test", "w") as f:
    for idx in range(len(y_test)):
        write_str = "%s\n" % ("\t".join([str(i) for i in y_test[idx]]))
        f.write(write_str)
with open("../data/people_daily/y_valid", "w") as f:
    for idx in range(len(y_valid)):
        write_str = "%s\n" % ("\t".join([str(i) for i in y_valid[idx]]))
        f.write(write_str)
```

数据集的构建此处是自己重写的，主要有两个原因：

- 避免使用pandas。pandas虽然有很多操作比较方便，但是个人认为在数据量较大的环境下，IO流操作比pandas更加省内存，在一些操作下甚至可以达到常数级别的空间复杂度(读一条操作一条输出一条)
- 避免使用pkl。pkl要求bytes编码存储，但实际上python3下虽然解决了中文的问题，但是编码仍有坑。

模型部分

数据处理完了，就到了非常关键的模型部分，此处使用的tensorflow==1.12.0。

为了更好的讲述此部分内容，我打算分开，将核心部分的内容拿来讲解，具体的代码[点击阅读原文](#)即可找到。

```
def __init__(self, config):
    self.config = config

    # 三个待输入的数据
    self.input_x = tf.placeholder(
        tf.int32, [None, self.config.seq_length], name='input_x')
    self.input_y = tf.placeholder(
```



```
tf.int32, [None, self.config.seq_length], name='input_y')
self.keep_prob = tf.placeholder(tf.float32, name='keep_prob')

self.bilstm_crf()
```

我仍然喜欢将tf模型单独用一个类来表示，此处是初始化，这块还是比较简单的。

```
with tf.name_scope("embedding"):
    # embedding layer
    w2v_matrix = tf.get_variable(name="w2v_matrix", shape=[
        self.config.vocab_size, self.config.embedding_dim], dtype=tf.float32, initializer=t
    embedding_inputs = tf.nn.embedding_lookup(w2v_matrix, self.input_x)
    embedding_inputs = tf.nn.dropout(embedding_inputs, self.keep_prob)
```

embedding部分，我偷个懒，也是希望尝试一下，看看直接用一个未训练的词向量表来作为预训练词向量模型，然后通过训练迭代转化，查看该方式的结果是否会出现意外。

```
with tf.name_scope("BiLSTM"):
    # BiLSTM layer
    lstm_fw_cell = tf.nn.rnn_cell.LSTMCell(
        100, forget_bias=1.0, state_is_tuple=True)
    lstm_bw_cell = tf.nn.rnn_cell.LSTMCell(
        100, forget_bias=1.0, state_is_tuple=True)
    (output_fw, output_bw), states = tf.nn.bidirectional_dynamic_rnn(lstm_fw_cell,
                                                                    lstm_bw_cell,
                                                                    embedding_inputs,
                                                                    dtype=tf.float32,
                                                                    time_major=False,
                                                                    scope=None)

    bilstm_out = tf.concat([output_fw, output_bw], axis=2)
    self.bilstm_tmp = bilstm_out
```

BiLSTM部分，其实非常简洁明了，这是一个双向LSTM的标准格式，就直接照搬啦，节点个数我是随便设置的，在我的测试看来，对结果似乎没有质的变化。

```
with tf.name_scope("dense"):
    W = tf.get_variable(name="W_dense", shape=[self.config.batch_size, 2 * 100, self.config
        dtype=tf.float32, initializer=tf.truncated_normal_initializer())
    b = tf.get_variable(name="b_dense", shape=[self.config.batch_size, self.config.seq_leng
        initializer=tf.zeros_initializer())
    dense_out = tf.tanh(tf.matmul(bilstm_out, W) + b)
```

按照计划应该是要进入CRF层了，但是由于CRF的输入必须与输出的维数一致，所以需要经过一个全连接层转化。

```

with tf.name_scope("crf"):
    # CRF
    sequence_lengths = np.full(
        self.config.batch_size, self.config.seq_length, dtype=np.int32)
    self.shape1 = sequence_lengths
    log_likelihood, self.transition_params = tf.contrib.crf.crf_log_likelihood(
        dense_out, self.input_y, sequence_lengths)
    self.viterbi_sequence, self.viterbi_score = tf.contrib.crf.crf_decode(
        dense_out, self.transition_params, sequence_lengths)

```

tensorflow提供CRF的接口，仔细看看文档，查阅tensorflow的API文档就会发现，tf.contrib下有crf的层级结构，甚至看到里面常见的函数，另外，还推荐大家看一个其实非常有用但是大家似乎都不太愿意看的东西——源码，这块的坑我是通过看源码以及内部的注释最后爬出来的，这块的API其实很多版本都会不同，所以非常建议大家好好看看自己版本下这块代码是怎么写的。

下面是我看到的有关crf_log_likelihood下的代码：

```

def crf_log_likelihood(inputs,
                       tag_indices,
                       sequence_lengths,
                       transition_params=None):
    """Computes the log-likelihood of tag sequences in a CRF.

    Args:
        inputs: A [batch_size, max_seq_len, num_tags] tensor of unary potentials
            to use as input to the CRF layer.
        tag_indices: A [batch_size, max_seq_len] matrix of tag indices for which we
            compute the log-likelihood.
        sequence_lengths: A [batch_size] vector of true sequence lengths.
        transition_params: A [num_tags, num_tags] transition matrix, if available.
    Returns:
        log_likelihood: A [batch_size] `Tensor` containing the log-likelihood of
            each example, given the sequence of tag indices.
        transition_params: A [num_tags, num_tags] transition matrix. This is either
            provided by the caller or created in this function.
    """
    # Get shape information.
    num_tags = inputs.get_shape()[2].value

    # Get the transition matrix if not provided.
    if transition_params is None:
        transition_params = vs.get_variable("transitions", [num_tags, num_tags])

    sequence_scores = crf_sequence_score(inputs, tag_indices, sequence_lengths,
                                          transition_params)
    log_norm = crf_log_norm(inputs, sequence_lengths, transition_params)

    # Normalize the scores to get the log-likelihood per example.
    log_likelihood = sequence_scores - log_norm
    return log_likelihood, transition_params

```


在函数的定义下，其实给了很长串的注释，里面说了Args——参数以及Returns——输出结果的具体含义，甚至是矩阵结构，非常完善，要使用这个函数你要放里面放什么，已经告诉你了，就像给了你菜谱你只要准备好材料按照他的要求往里面放就好了，另一方面代码可以协助你追溯你的某个变量的转移和更新，方便你理解计算流程，甚至可以模仿提升，这就是多阅读API文档和源码的一大好处。

```
self.loss = tf.reduce_mean(-log_likelihood)
optimizer = tf.train.AdamOptimizer(self.config.learning_rate)
self.train = optimizer.minimize(self.loss)
```

当然不能忘记的是，要把模型训练的内容加上。

在这个函数后面，我自己写了个单测，比较简单，大家可以参考：

```
# 单测
input_x = [[0, 1, 2], [2, 3, 4]]
input_y = [[1, 1, 0], [2, 2, 1]]
model_config = modelConfig()
model_config.batch_size = 2
model_config.embedding_dim = 5
model_config.num_classes = 3
model_config.seq_length = 3
model_config.vocab_size = 5

model = BiLSTM_CRF(model_config)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # print(shape1)
    crf_out = sess.run([model.viterbi_sequence, model.viterbi_score], feed_dict={
        "input_x:0": input_x, "input_y:0": input_y, "keep_prob:0":
    loss_out = sess.run([model.loss], feed_dict={
        "input_x:0": input_x, "input_y:0": input_y, "keep_prob:0": model_config.keep_prob
    print(loss_out)
    print(crf_out)
    for i in range(500):
        sess.run(model.train, feed_dict={
            "input_x:0": input_x, "input_y:0": input_y, "keep_prob:0": model_config.keep_prob
        crf_out = sess.run([model.viterbi_sequence, model.viterbi_score], feed_dict={
            "input_x:0": input_x, "input_y:0": input_y, "keep_prob:0":
        loss_out = sess.run([model.loss], feed_dict={
            "input_x:0": input_x, "input_y:0": input_y, "keep_prob:0": model_config.keep_prob
        print(loss_out)
        print(crf_out)
```

训练主程序

模型建立完，当然就到整理数据开始训练了。

为了保证数据本身的可解释性，所以我没有在数据预处理阶段就把文字和标签转化为数字，而是留了词典，然后在训练主程序里面，读入原文本原标签，读入词典，然后来进行转化。

下面是加载数据和转化的示例。

```
# 数据加载
x_train = utils.load_dataset(x_train_PATH, pad_len=SEQ_LEN)
y_train = utils.load_dataset(y_train_PATH, pad_len=SEQ_LEN)

# 字典加载
t2id_dict = utils.load_2id_dic(t2id_PATH)
w2id_dict = utils.load_2id_dic(w2id_PATH)

# 数据转化
x_train = utils.item2id_batch(x_train, w2id_dict)
y_train = utils.item2id_batch(y_train, t2id_dict)
```

具体这几步的函数定义如下：

```
def load_dataset(path, batch_size=64, pad_len=30):
    dataset = []
    with open(path, encoding="utf8") as f:
        data_batch = []
        for line in f:
            ll = line.strip().split("\t")
            while len(ll) < pad_len:
                ll.append("")
            data_batch.append(ll[:pad_len])
            if len(data_batch) == batch_size:
                dataset.append(data_batch)
                data_batch = []
        return dataset

def load_2id_dic(path):
    dic_get = {}
    with open(path) as f:
        for line in f:
            ll = line.strip().split("\t")
            if len(ll) < 2:
                dic_get[""] = 0
            else:
                dic_get[ll[0]] = int(ll[1])
    return dic_get

def item2id_batch(items_batch, dic_get):
    res = []
    for batch_ in items_batch:
        res_batch = []
```

```

    for item in batch_:
        sentence = []
        for i in item:
            if i in dic_get:
                sentence.append(dic_get[i])
        res_batch.append(sentence)
    res.append(res_batch)
return res

```

然后就可以开始进行模型初始化了，另外还需要配置好一些必要的参数。

```

# 模型初始化
modelConf = modelConfig()
modelConf.seq_length = len(x_train[-1][-1]) # 序列长度
modelConf.num_classes = len(t2id_dict) # 类别数
modelConf.batch_size = len(x_train[-1]) # 每批训练大小
modelConf.num_batches = len(x_train) # 一共有多少batch
modelConf.vocab_size = len(w2id_dict) # 词汇量
modelConf.num_epochs = 10 # 迭代代数
model = BiLSTM_CRF(modelConf)

```

modelConfig是我自己写的一个有关超参数的类，里面有一些默认值，此处对一些需要修改的默认值进行更新，这个类的定义如下：

```

class modelConfig(object):
    """模型必要参数"""

    embedding_dim = 300 # 词向量维度
    seq_length = 20 # 序列长度
    num_classes = 11 # 类别数
    # hidden_dim = 64 # 全连接层神经元

    keep_prob = 0.5 # dropout保留比例
    learning_rate = 1e-4 # 学习率

    batch_size = 64 # 每批训练大小
    num_batches = 263 # 一共有多少batch
    num_epochs = 20 # 总迭代轮次

    print_per_batch = 100 # 每多少轮输出一次结果

```

后续就是激动人心的训练了，下面代码默认tf.Session()已经打开且经过了initialize。

```

tmp_batch_id = 0
# training
while tmp_batch_id < len(x_train):
    sess.run(model.train, feed_dict={
        "input_x:0": x_train[tmp_batch_id], "input_y:0": y_train[tmp_batch_id], "keep_prob:0"
    })
    tmp_batch_id = tmp_batch_id + 1
    loss = sess.run(model.loss, feed_dict={

```

```
"input_x:0": x_train[0], "input_y:0": y_train[0], "keep_prob:0": modelConf.keep_prob]
```

在一个epoch训练下，采用批量法，所以每个epoch下又有每个batch训练。

```
# validating
tmp_batch_id = 0
y_pred = []
y_valid_combine = []
while tmp_batch_id < len(x_valid):
    y_pred_batch = sess.run(model.viterbi_sequence, feed_dict={
        "input_x:0": x_valid[tmp_batch_id], "input_y:0": y_valid[tmp_batch_id], "keep_prob:0":
    for idx in range(len(y_pred_batch)):
        y_pred = y_pred + y_pred_batch[idx].tolist()
        y_valid_combine = y_valid_combine + y_valid[tmp_batch_id][idx]
        tmp_batch_id = tmp_batch_id + 1
    p, r, f1score = utils.model_rep(y_pred, y_valid_combine)
    print("epoch: %s, loss:%s, f1: %s" %
          (i, loss, f1score))
    utils.print_matrix(utils.model_conf(y_pred, y_valid_combine))
    print("-----")
```

然后为了检验每一代的结果，此处还进行了一次validate，打印了loss、F1和混淆矩阵。

这里自己写了几个工具函数，用于结果展示：

```
from sklearn.metrics import precision_score, recall_score, f1_score
def model_rep(y_true, y_pred, average="micro"):
    p = precision_score(y_true, y_pred, average=average)
    r = recall_score(y_true, y_pred, average=average)
    f1score = f1_score(y_true, y_pred, average=average)
    return p, r, f1score
```

首先是准确率、召回率和F1值。

```
from sklearn.metrics import confusion_matrix
def model_conf(y_true, y_pred):
    return confusion_matrix(y_true, y_pred)
```

然后是混淆矩阵(呃呃呃，似乎是有点画蛇添足，我就是想统一格式哈哈哈)。

最后是一个结果输出规范化的函数，主要针对二维数组(或者说矩阵)的输出格式化，可以避免无语的方括号以及不必要的换行。

```
def print_matrix(mat):
    for idx in range(len(mat)):
        for j in mat[idx]:
```

```
print("%s\t" % j, end="")
print("\n", end="")
print("", end="\n")
```

存在缺陷

结果是跑通了哈哈哈哈哈，但是，在写代码的过程中和结果评估的过程中，其实发现了一些问题，后续需要改进，空间可能还不小，我在这里抛砖引玉，如果有更好的方案欢迎大家提出。

- 在进行预测阶段，仍需要凑够batch_size个才能够进行预测，不能一个一个预测，主要原因在于条件随机场计算下输入矩阵有要求。
- 虽然评价指标数据都非常好看，但是看了混淆矩阵就会发现并不理想，原因在于padding阶段补长策略下，以及实际问题下，补充标签和无属性的点过多，样本极度不平衡。

后记

命名实体识别是我感觉未来需要用到，所以尝试开始入门，这是我写的第一个这方面的程序，第一次总是比较痛苦，写这块花了点时间和代价，但是收获慢慢，自己无论是技术上还是理论上都有不少提升，感觉很好，也不枉我看了这么多论文、博客、文档、源码，写了这么多代码了啊啊啊。所以还是强调，非常建议大家多去看文章，多动手，想要提升，没有捷径，只有不断的刻苦学习和练习，道理都懂，但是最终能到达终点的终究是少数，理由就在于成功的人真的一直在践行。

我是叉烧，欢迎关注我！

叉烧，机器学习算法实习生，北京科技大学数理学院统计学研二硕士毕业，本科北京科技大学信息与计算科学、金融工程双学位毕业，硕士期间发表论文6篇，学生一作3篇，1项国家自然科学基金面上项目学生第2参与者，参与国家级及以上学术会议4次，其中，1次优秀论文，国家奖学金，北京市优秀毕业生。曾任去哪儿网大住宿事业部产品数据，美团点评出行事业部算法工程师。



微信个人公众号
CS的陋室

微信 zgr950123
邮箱 chashaozgr@163.com
知乎 机智的叉烧

[阅读原文](#)