《自然语言处理》
-词法分析

华为技术有限公司

目录

1 实验总览

1.1 实验简介

词性标注是自然语言处理的一个应用，是给单词指定标签的过程。本实验主要介绍条件随机场(Conditional Random Field，CRF)在词性标注方面的应用。CRF 是给定一组输入随机变量条件下另一组输出随机变量的条件概率分布模型，其特点是假设输出随机变量构成马尔科夫随机场。条件随机场可以用于不同的预测问题，这时主要使用的线性链条件随机场，将线性链条件随机场应用于标注问题是 Lafferty 等人在 2001 年提出的。线性链条件随机场中的状态变量形成一个线性链，类似于数据结构中的链表结构，每个节点只与前一个节点（如果存在）及后一个节点（如果存在）有关，即在时间序列中每个变量只和前一时刻、后一个时刻的变量有关：

$$P(y_i|\mathrm{x}, y_1, \ldots, y_n) = P(y_i|x, y_{i-1}, y_i)$$

本章实验难度分为中级和高级。

中级实验：CRF 算法代码实现；

高级实验：CRF 实现词性标注；

1.2 实验目的

本章的主要内容分为两块，一块是基于 Python3.6 及 mindspore 深度学习框架实现 CRF 算法，另一块基于 python3.6 以及 keras 训练 bi-lstm，然后结合 CRF 实现词性标注，主要目的是为了让学员知道什么是 CRF、CRF 的作用以及如何使用 CRF 进行词性标注。

1.3 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

实验	简述	难度	软件环境	开发环境

CRF 算法代码实现	使用 mindspore 框架实现 CRF 算法	中级	Python3.6、mindspore0.5	PC 64bit
CRF 实现词性标注	基于 Python3.6 以及 keras 训练 bi-lstm，然后结合 CRF 来实现词性标注	高级	Python3.6、Tensorflow1.8	ModelArts

2 CRF 算法实现实验

2.1 实验简介

条件随机场(Conditional Random Field，CRF)是给定变量 X 条件下，随机变量 Y 的马尔可夫随机场。本实验主要介绍定义在线性链上的特殊的条件随机场，称为线性链条件随机场(liner chain conditional random field)，并通过深度学习框架 mindspore 实现线性链 CRF 算法。线性链条件随机场常用于标注等问题。这时，在条件概率模型 P(Y/X)中，Y 是输出变量，表示标记序列，X 是输入变量，表示需要标注的观测序列。学习时，利用训练数据集通过极大似然估计或正则化极大似然估计得到条件概率模型 $\bar{P}$ (Y/X)；预测时，对于给定的输入序列 x，求出条件概率 $\bar{P}$(y/x)最大的输出序列 $\bar{y}$。

【实验环境要求】:

1、python3.6

2、mindsporeo.5

3、本机

2.2 实验预备知识

本实验操作需要学员有 3 方面的预备知识做支撑：

- 有自然语言处理中概率模型，马尔科夫随机场等概念基础。
- 对深度学习框架 mindspore 有一定的应用基础。
- 对 CRF 理论有一定的了解。

2.3 理论介绍

2.3.1 条件随机场简介

机器学习最重要的任务，是根据一些已观察到的证据（例如训练样本）来对感兴趣的未知变量（例如类别标记）进行估计和推测。1

概率模型提供这样一种描述的框架，将学习任务归结于计算变量的概率分布。在概率模型中，利用已知变量推测未知变量的分布称为“推断”，其核心是如何基于可观测变量推测出未知变量的条件分布。具体来说，假定所关心的变量集合为 Y，可观测变量集合为 X，“生成式”模型直接通过训练样本基本联合概率分布 P(Y,X)；“判别式”模型通过先计算条件分布 P(Y|X)。

HMM 是一种生成式概率图模型，条件随机场（CRF）与 HMM 不同，是一种判别式的概率图模型。CRF 是在给定一组变量的情况下，求解另一组变量的条件概率的模型。

设 X 与 Y 是一组随机变量，P(Y,X)是给定随机变量 X 情况下，随机变量 Y 的条件概率。若随机变量 Y 构成一个无向图 G(V,E)，当 X 与 Y 两个随机变量的概率分布满足如下的条件：

$$P(Y_v \mid X, Y_{V/\{v\}}) = P(Y_v \mid X, Y_{n(v)})$$

图 2-1 概率分布条件

则称在给定随机变量序列 X 的情况下，随机变量序列 Y 的条件概率 P(Y,X)构成条件随机场。

简单说明一下上面的条件概率公式：

v 表示 G 中的任一节点，例如 Y1，v~V。n(v)表示与 v 有边连接的节点的集合。上式的含义就是，Y 在 i 时刻的状态，仅与其有边连接的节点有关。

在 NLP 中，常用的是线性链的条件随机场，下面着重介绍下线性链条件随机场以加深理解。

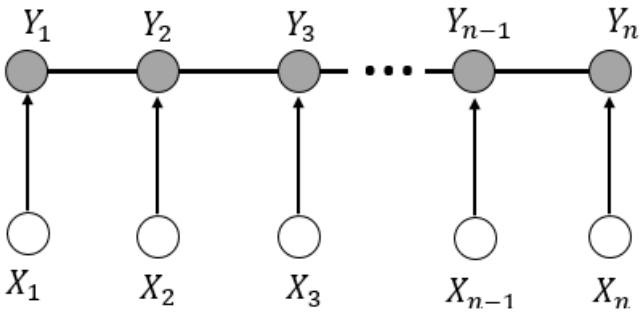


图 2-1 线性链条件随机场

设 X={x1,x2,x3,....xn}，Y={y1,y2,y3,....yn}均为线性链表示的随机变量序列，若在给定随机变量序列 X 的情况下，随机变量序列 Y 的条件概率 P(Y,X)构成条件随机场，即满足如下的条件：

$$\mathrm{P(yi|X, y1, y2, \ldots yi-1, yi+1, \ldots yn) = P(yi|X, yi-1, yi+1)}$$

从上面的定义可以看出，条件随机场与 HMM 之间的差异在于，HMM 中，Y 在 i 时刻状态与其前一时刻，即 y(i-1)相关。而在 CRF 中，Y 在 i 时刻的状态与其前后时刻，即 y(i-1)与 y(i+1)均相关。

在介绍 CRF 的实际应用之前，还有一些概念需要介绍，就是条件随机场的参数化形式。

2.3.2 条件随机场的参数化表现形式

我们先列出来 CRF 的参数化形式。假设 P(Y,X)是随机序列 Y 在给定随机序列 X 情况下的条件随机场，则在随机变量 X 取值为 x 的情况下，随机变量 Y 的取值 y 具有如下关系：

$$P(\mathrm{y} \mid x) = \frac{1}{Z(x)} \exp\left(\sum_{i,k} v_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} u_l s_l(y_i, x, i)\right)$$

式中

$$Z(x) = \sum_{\mathrm{y}} \exp\left(\sum_{i,k} v_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,l} u_l s_l(y_i, x, i)\right)$$

图 2-3 线性链条件随机场的参数化表现形式

t_k 和 s_l 是特征函数，v_k 和 u_l 是对应的权值

t_k 是状态转移函数，v_k 是对应的权值；s_l 是发射函数，u_l 是对应的权值。假如所有的 t_k，s_l 和 v_k，u_l 都已知，我们要算的 P(Yi =yi|X)就可以计算。

在给定随机序列 X 的情况下，计算概率最大 Y 序列可以用维特比算法。

tk，sl 和 vk，ul 如何确定和学习？下面让我们通过代码来进一步学习。

2.4 代码实现

2.4.1 导入相应的包

```
'''
CRF script.
'''
import numpy as np
import mindspore.nn as nn
from mindspore.ops import operations as P
from mindspore.common.tensor import Tensor
from mindspore.common.parameter import Parameter
import mindspore.common.dtype as mstype
```

2.4.2 CRF 类的创建

```
class CRF(nn.Cell):
```

2.4.3 参数说明

```
    '''
    Conditional Random Field
    Args:
        tag_to_index: The dict for tag to index mapping with extra "<START>" and "<STOP>"sign.
        batch_size: Batch size, i.e., the length of the first dimension.
        seq_length: Sequence length, i.e., the length of the second dimention.
        is_training: Specifies whether to use training mode.
    Returns:
        Training mode: Tensor, total loss.
        Evaluation mode: Tuple, the index for each step with the highest score; Tuple, the index for the last step
with the highest score.
    '''
```

2.4.4 定义__init__方法

```
    def __init__(self, tag_to_index, batch_size=1, seq_length=128, is_training=True):
        super(CRF, self).__init__()
        self.target_size = len(tag_to_index)
```

```
        self.is_training = is_training
        self.tag_to_index = tag_to_index
        self.batch_size = batch_size
        self.seq_length = seq_length
        self.START_TAG = "<START>"
        self.STOP_TAG = "<STOP>"
        self.START_VALUE = Tensor(self.target_size-2, dtype=mstype.int32)
        self.STOP_VALUE = Tensor(self.target_size-1, dtype=mstype.int32)
        #定义转换矩阵
        transitions = np.random.normal(size=(self.target_size, self.target_size)).astype(np.float32)
        #将开始标记和结尾标记设置为较小值，标注是不易被选择(符合实际情况)
        transitions[tag_to_index[self.START_TAG], :] = -10000
        transitions[:, tag_to_index[self.STOP_TAG]] = -10000
       #使用 Parameter 方法
        self.transitions = Parameter(Tensor(transitions), name="transition_matrix")
       #使用 P 方法
        self.cat = P.Concat(axis=-1)
        self.argmax = P.ArgMaxWithValue(axis=-1)
        self.log = P.Log()
        self.exp = P.Exp()
        self.sum = P.ReduceSum()
        self.tile = P.Tile()
        self.reduce_sum = P.ReduceSum(keep_dims=True)
        self.reshape = P.Reshape()
        self.expand = P.ExpandDims()
        self.mean = P.ReduceMean()
        init_alphas = np.ones(shape=(self.batch_size, self.target_size)) * -10000.0
        init_alphas[:, self.tag_to_index[self.START_TAG]] = 0.
        self.init_alphas = Tensor(init_alphas, dtype=mstype.float32)
        self.cast = P.Cast()
        self.reduce_max = P.ReduceMax(keep_dims=True)
        self.on_value = Tensor(1.0, dtype=mstype.float32)
        self.off_value = Tensor(0.0, dtype=mstype.float32)
        self.onehot = P.OneHot()
```

2.4.5 计算归一化因子的 log_sum_exp 分数

```
    def log_sum_exp(self, logits):
        '''
        Compute the log_sum_exp score for normalization factor.
        '''
        max_score = self.reduce_max(logits, -1)     #16   5   5
        score = self.log(self.reduce_sum(self.exp(logits - max_score), -1))
        score = max_score + score
        return score
```

2.4.6 定义_realpath_score 方法，计算真实路径的放射和转移分数

```
    def _realpath_score(self, features, label):
        '''
        Compute the emission and transition score for the real path.
        '''
        label = label * 1
        concat_A = self.tile(self.reshape(self.START_VALUE, (1,)), (self.batch_size,))
        concat_A = self.reshape(concat_A, (self.batch_size, 1))
        labels = self.cat((concat_A, label))
        onehot_label = self.onehot(label, self.target_size, self.on_value, self.off_value)
        emits = features * onehot_label
        labels = self.onehot(labels, self.target_size, self.on_value, self.off_value)
        label1 = labels[:, 1:, :]
        label2 = labels[:, :self.seq_length, :]
        label1 = self.expand(label1, 3)
        label2 = self.expand(label2, 2)
        label_trans = label1 * label2
        transitions = self.expand(self.expand(self.transitions, 0), 0)
        trans = transitions * label_trans
        score = self.sum(emits, (1, 2)) + self.sum(trans, (1, 2, 3))
        stop_value_index = labels[:, (self.seq_length-1):self.seq_length, :]
        stop_value = self.transitions[(self.target_size-1):self.target_size, :]
        stop_score = stop_value * self.reshape(stop_value_index, (self.batch_size, self.target_size))
        score = score + self.sum(stop_score, 1)
        score = self.reshape(score, (self.batch_size, -1))
        return score
```

2.4.7 定义_normalization_factor()方法，计算所有可能的输出序列的分数

```
    def _normalization_factor(self, features):
        '''
        Compute the total score for all the paths.
        '''
        forward_var = self.init_alphas
        forward_var = self.expand(forward_var, 1)
        for idx in range(self.seq_length):
            feat = features[:, idx:(idx+1), :]
            emit_score = self.reshape(feat, (self.batch_size, self.target_size, 1))
            next_tag_var = emit_score + self.transitions + forward_var
            forward_var = self.log_sum_exp(next_tag_var)
            forward_var = self.reshape(forward_var, (self.batch_size, 1, self.target_size))
        terminal_var = forward_var + self.reshape(self.transitions[(self.target_size-1):self.target_size, :],
(1, -1))
        alpha = self.log_sum_exp(terminal_var)
        alpha = self.reshape(alpha, (self.batch_size, -1))
```

```
            return alpha
```

2.4.8 定义_decoder()方法，用维特比算法解码概率最大的 Y 序列

```
    def _decoder(self, features):
        '''
        Viterbi decode for evaluation.
        '''
        backpointers = ()
        forward_var = self.init_alphas
        for idx in range(self.seq_length):
            feat = features[:, idx:(idx+1), :]
            feat = self.reshape(feat, (self.batch_size, self.target_size))
            bptrs_t = ()

            next_tag_var = self.expand(forward_var, 1) + self.transitions
            best_tag_id, best_tag_value = self.argmax(next_tag_var)
            bptrs_t += (best_tag_id,)
            forward_var = best_tag_value + feat

            backpointers += (bptrs_t,)
        terminal_var = forward_var + self.reshape(self.transitions[(self.target_size-1):self.target_size, :],
(1, -1))
        best_tag_id, _ = self.argmax(terminal_var)
        return backpointers, best_tag_id
```

2.4.9 定义 construct()函数，更新训练和验证环节的路径及对应的标签

```
    def construct(self, features, label):
        if self.is_training:
            forward_score = self._normalization_factor(features)
            gold_score = self._realpath_score(features, label)
            return_value = self.mean(forward_score - gold_score)
        else:
            path_list, tag = self._decoder(features)
            return_value = path_list, tag
        return return_value
```

2.4.10 定义 postprocess()函数，进行后处理，找出最佳路径

```
def postprocess(backpointers, best_tag_id):
    '''
    Do postprocess
    '''
    best_tag_id = best_tag_id.asnumpy()
    batch_size = len(best_tag_id)
    best_path = []
    for i in range(batch_size):
        best_path.append([])
```

```
            best_local_id = best_tag_id[i]
            best_path[-1].append(best_local_id)
            for bptrs_t in reversed(backpointers):
                bptrs_t = bptrs_t[0].asnumpy()
                local_idx = bptrs_t[i]
                best_local_id = local_idx[best_local_id]
                best_path[-1].append(best_local_id)
            # Pop off the start tag (we dont want to return that to the caller)
            best_path[-1].pop()
            best_path[-1].reverse()
    return best_path
```

3 CRF 实现词性标注实验

3.1 实验简介

本实验将展示 CRF 在词性标注方面的应用。

3.2 实验目的

本章的主要内容是基于 Python3.6 以及 keras 训练 bi-lstm，然后结合 CRF 来实现词性标注的，本实验是为了让学员知道 CRF 的作用以及如何使用 CRF 进行词性标注。

3.3 实验预备知识

本实验操作需要学员有 2 方面的预备知识做支撑：

- 有相应 Python 语言编程基础，能够按照实验步骤安装包和下载相应的数据。
- 有 Keras 的操作基础。

3.4 实验环境

本实验可以在 ModelArts 平台进行，也可以在本机进行，具体环境准备如下：

在 ModelArts 平台进行实验时，需要提前将数据 people-2014.zip 以及安装包 keras-contrib-master.zip 和 bi-lstm-crf-master.zip 上传，新建 notebook 时进行关联对应的 OBS 文件路径，然后进行数据预处理和包的安装，具体参见实验步骤。

在本机进行实验时，将数据 people-2014.zip 以及安装包 keras-contrib-master.zip 和 bi-lstm-crf-master.zip 放于一个文件夹，创建 notebook 进行数据预处理和包的安装，具体参见实验步骤。

3.5 实验步骤

3.5.1 安装包

1）安装 keras-preprocessing

```
!pip install keras-preprocessing==1.0.9
```

```
Collecting keras-preprocessing==1.0.9
  Downloading http://repo.myhuaweicloud.com/repository/pypi/packages/c0/bf/0315ef6a9fd3fc2346e85b0ff1f5f83ca17073f2c31ac719ab2e4da0d4a3/Keras_Preprocessing-1.0.9-py2.py3-none-any.whl (59kB)
    100% |████████████████████████████████| 61kB 27.4MB/s ta 0:00:01
Requirement already satisfied: numpy>=1.9.1 in /opt/conda/envs/python36_tf/lib/python3.6/site-packages (from keras-preprocessing==1.0.9)
Requirement already satisfied: six>=1.9.0 in ./modelarts-sdk (from keras-preprocessing==1.0.9)
Installing collected packages: keras-preprocessing
  Found existing installation: Keras-Preprocessing 1.0.1
    Uninstalling Keras-Preprocessing-1.0.1:
      Successfully uninstalled Keras-Preprocessing-1.0.1
Successfully installed keras-preprocessing-1.0.9
You are using pip version 9.0.1, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

图 2-4 安装 keras-preprocessing

2）解压 keras-contrib-master.zip

!unzip work/keras-contrib-master.zip #解压安装包

```
In [1]: !unzip work/keras-contrib-master.zip
```

```
  creating: keras-contrib-master/tests/keras_contrib/metrics/
extracting: keras-contrib-master/tests/keras_contrib/metrics/.gitkeep
  creating: keras-contrib-master/tests/keras_contrib/optimizers/
 inflating: keras-contrib-master/tests/keras_contrib/optimizers/ftml_test.py
 inflating: keras-contrib-master/tests/keras_contrib/optimizers/lars_test.py
 inflating: keras-contrib-master/tests/keras_contrib/optimizers/padam_test.py
 inflating: keras-contrib-master/tests/keras_contrib/optimizers/yogi_test.py
  creating: keras-contrib-master/tests/keras_contrib/preprocessing/
extracting: keras-contrib-master/tests/keras_contrib/preprocessing/.gitkeep
  creating: keras-contrib-master/tests/keras_contrib/regularizers/
extracting: keras-contrib-master/tests/keras_contrib/regularizers/.gitkeep
  creating: keras-contrib-master/tests/keras_contrib/utils/
 inflating: keras-contrib-master/tests/keras_contrib/utils/save_load_utils_test.py
  creating: keras-contrib-master/tests/keras_contrib/wrappers/
extracting: keras-contrib-master/tests/keras_contrib/wrappers/.gitkeep
  creating: keras-contrib-master/tests/tooling/
 inflating: keras-contrib-master/tests/tooling/test_codeowners.py
 inflating: keras-contrib-master/tests/tooling/test_doc_auto_generation.py
 inflating: keras-contrib-master/tests/tooling/test_documentation.py
```

图 2-5 解压 keras-contrib-master.zip

3）安装 keras-contrib-master

```
!python keras-contrib-master/setup.py install
```

```
!python keras-contrib-master/setup.py install
```

```
Adding six 1.12.0 to easy-install.pth file

Using /home/jovyan/modelarts-sdk
Searching for PyYAML==3.12
Best match: PyYAML 3.12
Adding PyYAML 3.12 to easy-install.pth file

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages
Searching for Keras-Applications==1.0.2
Best match: Keras-Applications 1.0.2
Adding Keras-Applications 1.0.2 to easy-install.pth file

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages
Searching for scipy==1.0.0
Best match: scipy 1.0.0
Adding scipy 1.0.0 to easy-install.pth file

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages
Finished processing dependencies for keras-contrib==2.0.8
```

图 2-6 安装 keras-contrib-master

4）解压安装包 bi-lstm-crf-master.zip

```
!unzip work/bi-lstm-crf-master.zip #解压安装包
```

```
!unzip work/bi-lstm-crf-master.zip
  inflating: bi-lstm-crf-master/logs/events.out.tfevents.1561372945.DESKTOP-JFDDI3B
  inflating: bi-lstm-crf-master/logs/events.out.tfevents.1561375018.DESKTOP-JFDDI3B
  inflating: bi-lstm-crf-master/logs/events.out.tfevents.1561375982.DESKTOP-JFDDI3B
   creating: bi-lstm-crf-master/models/
  inflating: bi-lstm-crf-master/models/weights.12--0.03.h5
  inflating: bi-lstm-crf-master/models/weights.32--0.18.h5
  inflating: bi-lstm-crf-master/setup - 副本.py
  inflating: bi-lstm-crf-master/setup.py
   creating: bi-lstm-crf-master/tools/
  inflating: bi-lstm-crf-master/tools/convert_to_h5.py
  inflating: bi-lstm-crf-master/tools/make_dicts.py
  inflating: bi-lstm-crf-master/tools/ner_data_preprocess.py
  inflating: bi-lstm-crf-master/tools/predict.py
  inflating: bi-lstm-crf-master/tools/score.py
  inflating: bi-lstm-crf-master/tools/score_preprocess.py
  inflating: bi-lstm-crf-master/tools/total_size.py
  inflating: bi-lstm-crf-master/tools/word_count.py
 extracting: bi-lstm-crf-master/tools/__init__.py
  inflating: bi-lstm-crf-master/train_example.py
```

图 2-7 解压 bi-lstm-crf-master.zip

5）安装 bi-lstm-crf

```
!python bi-lstm-crf-master/setup.py install
```

```
!python bi-lstm-crf-master/setup.py install
Keras-Preprocessing 1.0.1 is already the active version in easy-install.pth

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages/Keras_Preprocessing-1.0.1-py3.6.egg
Searching for h5py==2.8.0
Best match: h5py 2.8.0
Adding h5py 2.8.0 to easy-install.pth file

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages
Searching for numpy==1.14.5
Best match: numpy 1.14.5
Adding numpy 1.14.5 to easy-install.pth file

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages
Searching for scipy==1.0.0
Best match: scipy 1.0.0
Adding scipy 1.0.0 to easy-install.pth file

Using /opt/conda/envs/python36_tf/lib/python3.6/site-packages
Finished processing dependencies for dl-segmenter===0.1-SNAPSHOT
```

图 2-8 安装 bi-lstm-crf

3.5.2 配置环境，并导入包

```
import sys
path1 = "/home/jovyan/bi-lstm-crf-master"        #根据自己解压的文件路径添加
sys.path.append(path1)
path2 = "/home/jovyan/keras-contrib-master"    #根据自己解压的文件路径添加
sys.path.append(path2)
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.optimizers import Adam
```

```
from dl_segmenter import get_or_create, save_config,DLSegmenter
from dl_segmenter.custom.callbacks import LRFinder, SGDRScheduler, WatchScheduler
from dl_segmenter.data_loader import DataLoader
from dl_segmenter.utils import make_dictionaries
import os
import re
```

如果步骤 1 都成功了，但是此处仍然提示没有模块 dl_segmenter，请参考步骤 8 中 1）的方法重启 kernel 后，再执行这一步骤。

3.5.3 解压数据

```
!unzip work/people-2014.zip
```

```
!unzip work/people-2014.zip
  inflating: people-2014/train/0123/c82840-20456209.txt
  inflating: people-2014/train/0123/c82840-20456360.txt
  inflating: people-2014/train/0123/c82840-20456373.txt
  inflating: people-2014/train/0123/c82840-20457338.txt
  inflating: people-2014/train/0123/c82840-20457359.txt
  inflating: people-2014/train/0123/c82840-20457362.txt
  inflating: people-2014/train/0123/c82840-20457858.txt
  inflating: people-2014/train/0123/c82840-20461689.txt
  inflating: people-2014/train/0123/c82840-20461762.txt
  inflating: people-2014/train/0123/c82840-20461855.txt
  inflating: people-2014/train/0123/c82840-20461945.txt
  inflating: people-2014/train/0123/c82840-20462145.txt
  inflating: people-2014/train/0123/c82840-20462156.txt
  inflating: people-2014/train/0123/c82840-20462529.txt
  inflating: people-2014/train/0123/c82846-20457747.txt
  inflating: people-2014/train/0123/c82846-20458249.txt
  inflating: people-2014/train/0123/c82846-20458287.txt
  inflating: people-2014/train/0123/c85914-24209618.txt
  inflating: people-2014/train/0123/c87423-24203635.txt
```

图 2-9 解压需要使用的数据

3.5.4 数据预处理

1） 将标注的语料转化成 BIS 形式，并合并在一个文件中

```
#B:表示语句块的开始，I:表示非语句块的开始，S:表示单独成词
#示例如下：
# 华尔街/nsf 股市/n 。/w    ->    华 尔 街 股 市 。   B-nsf  I-nsf  I-nsf  B-n  I-n  S-w
def print_process(process):
    num_processed = int(30 * process)
    num_unprocessed = 30 - num_processed
    print(
        f"{''.join(['['] + ['='] * num_processed + ['>'] + [' '] * num_unprocessed + [']'])}, {(process * 100):.2f} %")
```

```
def convert_to_bis(source_dir, target_path, log=False, combine=False, single_line=True):
    print("开始转化...")
    for root, dirs, files in os.walk(source_dir):
        total = len(files)
        tgt_dir = target_path + root[len(source_dir):]

        print(tgt_dir)
        for index, name in enumerate(files):
            file = os.path.join(root, name)
            bises = process_file(file)
if combine:
                _save_bises(bises, target_path, write_mode='a', single_line=single_line)
            else:
                os.makedirs(tgt_dir, exist_ok=True)
                _save_bises(bises, os.path.join(tgt_dir, name), single_line=single_line)
            if log:
                print_process((index + 1) / total)
    print("转化完成")


def _save_bises(bises, path, write_mode='w+', single_line=True):
    with open(path, mode=write_mode, encoding='UTF-8') as f:
        if single_line:
            for bis in bises:
                sent, tags = [], []
                for char, tag in bis:
                    sent.append(char)
                    tags.append(tag)
                sent = ' '.join(sent)
                tags = ' '.join(tags)
                f.write(sent + "\t" + tags)
                f.write('\n')
        else:
            for bis in bises:
                for char, tag in bis:
                    f.write(char + "\t" + tag + "\n")
                f.write("\n")

def process_file(file):
    with open(file, 'r', encoding='UTF-8') as f:
        text = f.readlines()
        bises = _parse_text(text)
    return bises


def _parse_text(text: list):
    bises = []
```

```
    for line in text:
        # remove POS tag
        line, _ = re.subn('\\n', '', line)
        if line == '' or line == '\n':
            continue
        words = re.split('\s+', line)

        if len(words) > MAX_LEN_SIZE:
            texts = re.split('[。？！，.?!,]/w', line)
            if len(min(texts, key=len)) > MAX_LEN_SIZE:
                continue
            bises.extend(_parse_text(texts))
        else:
            bises.append(_tag(words))
    return bises

#给指定的一行文本打上 BIS 标签
def _tag(words):
    bis = []
    # words = list(map(list, words))
    pre_word = None
    for word in words:
        pos_t = None
        tokens = word.split('/')
        if len(tokens) == 2:
            word, pos = tokens
        elif len(tokens) == 3:
            word, pos_t, pos = tokens
        else:
            continue

        word = list(word)
        pos = pos.upper()

        if len(word) == 0:
            continue
        if word[0] == '[':
            pre_word = word
            continue
        if pre_word is not None:
            pre_word += word
            if pos_t is None:
                continue
            elif pos_t[-1] != ']':
                continue
            else:
                word = pre_word[1:]
```

```
                    pre_word = None

            if len(word) == 1:
                bis.append((word[0], 'S-' + pos))
            else:
                for i, char in enumerate(word):
                    if i == 0:
                        bis.append((char, 'B-' + pos))
                    else:
                        bis.append((char, 'I-' + pos))
        # bis.append(('\n', 'O'))
        return bis

corups_dir="people-2014/train"                    #指定存放语料库的文件夹，程序将会递归查找目录下的文件。
output_path="bi-lstm-crf-master/data/2014_processed"    #指定标记好的文件的输出路径。
combine = True

#处理后的最大语句长度（将原句子按标点符号断句，若断句后的长度仍比最大长度长，将忽略）
MAX_LEN_SIZE = 150

log = False
convert_to_bis(corups_dir, output_path, log, combine)
```

输出打印如下：

```
开始转化...
bi-lstm-crf-master/data/2014_processed
bi-lstm-crf-master/data/2014_processed/0101
bi-lstm-crf-master/data/2014_processed/0102
bi-lstm-crf-master/data/2014_processed/0103
bi-lstm-crf-master/data/2014_processed/0104
bi-lstm-crf-master/data/2014_processed/0105
bi-lstm-crf-master/data/2014_processed/0106
bi-lstm-crf-master/data/2014_processed/0107
bi-lstm-crf-master/data/2014_processed/0108
bi-lstm-crf-master/data/2014_processed/0109
bi-lstm-crf-master/data/2014_processed/0110
bi-lstm-crf-master/data/2014_processed/0111
bi-lstm-crf-master/data/2014_processed/0112
bi-lstm-crf-master/data/2014_processed/0113
bi-lstm-crf-master/data/2014_processed/0114
bi-lstm-crf-master/data/2014_processed/0115
bi-lstm-crf-master/data/2014_processed/0116
bi-lstm-crf-master/data/2014_processed/0117
bi-lstm-crf-master/data/2014_processed/0118
bi-lstm-crf-master/data/2014_processed/0119
bi-lstm-crf-master/data/2014_processed/0120
bi-lstm-crf-master/data/2014_processed/0121
bi-lstm-crf-master/data/2014_processed/0122
bi-lstm-crf-master/data/2014_processed/0123
转化完成
```

图 2-10 语料转化过程

2） 生成字典

```
file_path = "bi-lstm-crf-master/data/2014_processed"        #用于生成字典的标注文件
src_dict_path = "bi-lstm-crf-master/config/src_dict.json"    #源字典保存路径
tgt_dict_path = "bi-lstm-crf-master/config/tgt_dict.json"    #目标字典保存路径
min_freq = 1                                                 #词频数阈值，小于该阈值的词将被忽略
print("开始生成...")
make_dictionaries(file_path,
                  src_dict_path=src_dict_path,
                  tgt_dict_path=tgt_dict_path,
                  filters="\t\n",
                  oov_token="<UNK>",
                  min_freq=min_freq)
print("生成字典结束.")
```

输出打印如下：

```
开始生成...
生成字典结束.
```

图 2-11 生成字典过程

3） 转化成 h5 文件

```
#可将文本文件 2014_processed 转换为 hdf5 格式，提升训练速度

src_dict_path = "bi-lstm-crf-master/config/src_dict.json"      #源字典保存路径
tgt_dict_path = "bi-lstm-crf-master/config/tgt_dict.json"      #目标字典保存路径
txt_path ="bi-lstm-crf-master/data/2014_processed"             #BIS 标注的文本文件路径
h5_path = "bi-lstm-crf-master/data/2014_processed.h5"          #转换为 hdf5 格式的保存路径
seq_len = 150                                                  #语句长度
data_loader = DataLoader(src_dict_path, tgt_dict_path,
                         batch_size=1,
                         max_len=seq_len,
                         sparse_target=False)
print("开始转化...")
data_loader.load_and_dump_to_h5(txt_path, h5_path, encoding='utf-8')
print("转化完成.")
```

输出结果打印：

```
开始转化...
转化完成.
```

图 2-12 转化 h5 文件过程

3.5.5 配置相关参数

```
h5_dataset_path = "bi-lstm-crf-master/data/2014_processed.h5"          #  转换为 hdf5 格式的数据集
config_save_path = "bi-lstm-crf-master/config/default-config.json"     #  模型配置路径
weights_save_path = "bi-lstm-crf-master/models/weights.{epoch:02d}-{val_loss:.2f}.h5"   #  模型权重保存路径
```

```
init_weights_path = "bi-lstm-crf-master/models/weights.32-0.18.sgdr.h5"   # 预训练模型权重文件路径
#embedding_file_path = "../data/sgns.renmin.word"                         # 词向量文件路径，若不使用设为 None
embedding_file_path = None                                                # 词向量文件路径，若不使用设为 None
src_dict_path = "bi-lstm-crf-master/config/src_dict.json"                 # 源字典路径
tgt_dict_path = "bi-lstm-crf-master/config/tgt_dict.json"                 # 目标字典路径
batch_size = 32
epochs = 10
# GPU 下用于选择训练的 GPU
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

data_loader = DataLoader(src_dict_path=src_dict_path,
                         tgt_dict_path=tgt_dict_path,
                         batch_size=batch_size)
steps_per_epoch = 2000
validation_steps = 20

config = {
    "vocab_size": data_loader.src_vocab_size,
    "chunk_size": data_loader.tgt_vocab_size,
    "embed_dim": 300,
    "bi_lstm_units": 256,
     "max_num_words": 20000,
    "dropout_rate": 0.1
}
```

3.5.6 加载数据

```
#加载数据,并将数据分割成训练集合验证集
X_train, Y_train, X_valid, Y_valid = DataLoader.load_data(h5_dataset_path, frac=0.8)
```

3.5.7 定义并训练模型

```
#定义模型
tokenizer = get_or_create(config,
                          optimizer=Adam(),
                          embedding_file=embedding_file_path,
                          src_dict_path=src_dict_path,
                          weights_path=init_weights_path)
print(tokenizer)
save_config(tokenizer, config_save_path)

#ModelCheckpoint 保存最佳模型
ck = ModelCheckpoint(weights_save_path,
                     save_best_only=True,
                     save_weights_only=True,
                     monitor='val_loss',
                     verbose=0)
#创建日志
```

```
log = TensorBoard(log_dir='../logs',
                  histogram_freq=0,
                  batch_size=data_loader.batch_size,
                  write_graph=True,
                  write_grads=False)

#  使用 LRFinder 寻找有效的学习率
lr_finder = LRFinder(1e-6, 1e-2, steps_per_epoch, epochs=1)                # => (2e-4, 3e-4)
lr_scheduler = WatchScheduler(lambda _, lr: lr / 2, min_lr=2e-4, max_lr=4e-4, watch="val_loss", watch_his_len=2)
lr_scheduler = SGDRScheduler(min_lr=4e-5, max_lr=1e-3, steps_per_epoch=steps_per_epoch,
                             cycle_length=15,
                             lr_decay=0.9,
                             mult_factor=1.2)
#训练模型
tokenizer.model.fit_generator(data_loader.generator_from_data(X_train, Y_train),
                              epochs=1,
                              steps_per_epoch=steps_per_epoch,
                              validation_data=data_loader.generator_from_data(X_valid, Y_valid),
                              validation_steps=validation_steps,
                              callbacks=[ck, log, lr_finder])
#画出损失函数
lr_finder.plot_loss()
```

输出结果：

```
<dl_segmenter.core.DLSegmenter object at 0x7f5819c92780>
Epoch 1/1
2000/2000 [==============================] - 17043s 9s/step - loss: 0.4510 - crf_accuracy: 0.8671 - val_loss: 0.1188 - val_crf_
accuracy: 0.9160
```

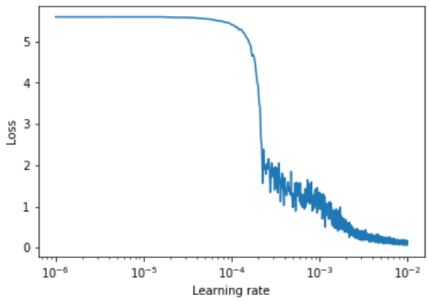


图 2-13 完成训练后的结果

3.5.8 词性标注测试

1）重启 kernel

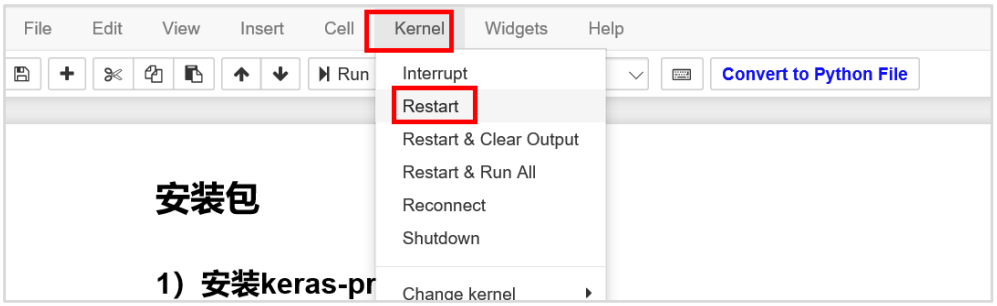


图 2-14 重启 kernel

2）导入包

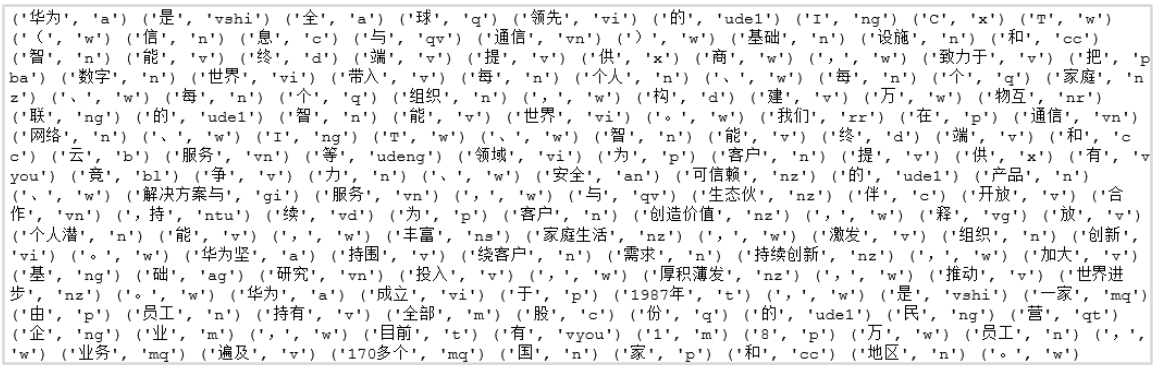
```
#配置环境，并导入包
import sys
path1 = "/home/jovyan/bi-lstm-crf-master"       #修改为自己的文件路径
sys.path.append(path1)
path2 = "/home/jovyan/keras-contrib-master"   #修改为自己的文件路径
sys.path.append(path2)
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.optimizers import Adam
from dl_segmenter import get_or_create, save_config,DLSegmenter
from dl_segmenter.custom.callbacks import LRFinder, SGDRScheduler, WatchScheduler
from dl_segmenter.data_loader import DataLoader
from dl_segmenter.utils import make_dictionaries
import os
import re
```

3）测试

```
segmenter: DLSegmenter = get_or_create("bi-lstm-crf-master/config/default-config.json",
                                       src_dict_path="bi-lstm-crf-master/config/src_dict.json",
                                       tgt_dict_path="bi-lstm-crf-master/config/tgt_dict.json",
                                       weights_path="bi-lstm-crf-master/models/weights.32--0.18.h5")
texts = [
    "华为是全球领先的 ICT（信息与通信）基础设施和智能终端提供商，"
     "致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界。"
     "我们在通信网络、IT、智能终端和云服务等领域为客户提供有竞争力、安全可信赖的产品、解决方案与服务，"
     "与生态伙伴开放合作，持续为客户创造价值，释放个人潜能，丰富家庭生活，激发组织创新。"
     "华为坚持围绕客户需求持续创新，加大基础研究投入，厚积薄发，推动世界进步。"
     "华为成立于 1987 年，是一家由员工持有全部股份的民营企业，目前有 18 万员工，业务遍及 170 多个国家和地区。"
]

for sent, tag in segmenter.decode_texts(texts):
      print(*zip(sent,tag))
```

结果：



```
('华为', 'a') ('是', 'vshi') ('全', 'a') ('球', 'q') ('领先', 'vi') ('的', 'ude1') ('I', 'ng') ('C', 'x') ('T', 'w')
('（', 'w') ('信', 'n') ('息', 'c') ('与', 'qv') ('通信', 'vn') ('）', 'w') ('基础', 'n') ('设施', 'n') ('和', 'cc')
('智', 'n') ('能', 'v') ('终', 'd') ('端', 'v') ('提', 'v') ('供', 'x') ('商', 'w') (',', 'w') ('致力于', 'v') ('把', 'p
ba') ('数字', 'n') ('世界', 'vi') ('带入', 'v') ('每', 'n') ('个人', 'n') ('、', 'w') ('每', 'n') ('个', 'q') ('家庭', 'n
z') ('、', 'w') ('每', 'n') ('个', 'q') ('组织', 'n') (',', 'w') ('构', 'd') ('建', 'v') ('万', 'w') ('物互', 'nr')
('联', 'ng') ('的', 'ude1') ('智', 'n') ('能', 'v') ('世界', 'vi') ('。', 'w') ('我们', 'rr') ('在', 'p') ('通信', 'vn')
('网络', 'n') ('、', 'w') ('I', 'ng') ('T', 'w') ('、', 'w') ('智', 'n') ('能', 'v') ('终', 'd') ('端', 'v') ('和', 'c
c') ('云', 'b') ('服务', 'vn') ('等', 'udeng') ('领域', 'vi') ('为', 'p') ('客户', 'n') ('提', 'v') ('供', 'x') ('有', 'v
you') ('竞', 'bl') ('争', 'v') ('力', 'n') ('、', 'w') ('安全', 'an') ('可信赖', 'nz') ('的', 'ude1') ('产品', 'n')
('、', 'w') ('解决方案与', 'gi') ('服务', 'vn') (',', 'w') ('与', 'qv') ('生态伙', 'nz') ('伴', 'c') ('开放', 'v') ('合
作', 'vn') (', 持', 'ntu') ('续', 'vd') ('为', 'p') ('客户', 'n') ('创造价值', 'nz') (',', 'w') ('释', 'vg') ('放', 'v')
('个人潜', 'n') ('能', 'v') (',', 'w') ('丰富', 'ns') ('家庭生活', 'nz') (',', 'w') ('激发', 'v') ('组织', 'n') ('创新',
'vi') ('。', 'w') ('华为坚', 'a') ('持围', 'v') ('绕客户', 'n') ('需求', 'n') ('持续创新', 'nz') (',', 'w') ('加大', 'v')
('基', 'ng') ('础', 'ag') ('研究', 'vn') ('投入', 'v') (',', 'w') ('厚积薄发', 'nz') (',', 'w') ('推动', 'v') ('世界进
步', 'nz') ('。', 'w') ('华为', 'a') ('成立', 'vi') ('于', 'p') ('1987年', 't') (',', 'w') ('是', 'vshi') ('一家', 'mq')
('由', 'p') ('员工', 'n') ('持有', 'v') ('全部', 'm') ('股', 'c') ('份', 'q') ('的', 'ude1') ('民', 'ng') ('营', 'qt')
('企', 'ng') ('业', 'm') (',', 'w') ('目前', 't') ('有', 'vyou') ('1', 'm') ('8', 'p') ('万', 'w') ('员工', 'n') (',',
'w') ('业务', 'mq') ('遍及', 'v') ('170多个', 'mq') ('国', 'n') ('家', 'p') ('和', 'cc') ('地区', 'n') ('。', 'w')
```

图 3-12 词性标注的结果

3.6 实验总结

本章的主要内容分为两块，一块基于深度学习框架 mindspore 完成了 CRF 算法的构建，另一块基于 Python3.6 以及 Keras 训练 bi-lstm，然后结合 CRF 来实现词性标注的，目的是是为了让学员知道什么是 CRF、CRF 的作用以及如何使用 CRF 进行词性标注。

3.7 开放题（可选）

自学 HMM 算法，总结比较 HMM 和 CRF 的不同点，并进一步思考，LSTM 依靠神经网络的超强的非线性拟合能力，完全可以胜任词性标注等序列标注问题，为什么后面要接 CRF 层?