《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 3](#_Toc48203766)

[1.1 实验简介 3](#_Toc48203767)

[1.2 实验目的 3](#_Toc48203768)

[1.3 实验清单 4](#_Toc48203769)

[2 CRF算法实现实验 5](#_Toc48203770)

[2.1 实验简介 5](#_Toc48203771)

[2.2 实验预备知识 5](#_Toc48203772)

[2.3 理论介绍 5](#_Toc48203773)

[2.3.1 条件随机场简介 5](#_Toc48203774)

[2.3.2 条件随机场的参数化表现形式 7](#_Toc48203775)

[2.4 代码实现 7](#_Toc48203776)

[2.4.1 导入相应的包 7](#_Toc48203777)

[2.4.2 CRF类的创建 7](#_Toc48203778)

[2.4.3 参数说明 7](#_Toc48203779)

[2.4.4 定义\_\_init\_\_方法 8](#_Toc48203780)

[2.4.5 计算归一化因子的log\_sum\_exp分数 9](#_Toc48203781)

[2.4.6 定义\_realpath\_score方法，计算真实路径的放射和转移分数 9](#_Toc48203782)

[2.4.7 定义\_normalization\_factor()方法，计算所有可能的输出序列的分数 9](#_Toc48203783)

[2.4.8 定义\_decoder()方法，用维特比算法解码概率最大的Y序列 10](#_Toc48203784)

[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 10](#_Toc48203785)

[2.4.10 定义postprocess()函数，进行后处理，找出最佳路径 11](#_Toc48203786)

[3 CRF实现词性标注实验 12](#_Toc48203787)

[3.1 实验简介 12](#_Toc48203788)

[3.2 实验目的 12](#_Toc48203789)

[3.3 实验预备知识 12](#_Toc48203790)

[3.4 实验环境 12](#_Toc48203791)

[3.5 实验步骤 13](#_Toc48203792)

[3.5.1 安装包 13](#_Toc48203793)

[3.5.2 配置环境，并导入包 15](#_Toc48203794)

[3.5.3 解压数据 15](#_Toc48203795)

[3.5.4 数据预处理 16](#_Toc48203796)

[3.5.5 配置相关参数 20](#_Toc48203797)

[3.5.6 加载数据 21](#_Toc48203798)

[3.5.7 定义并训练模型 21](#_Toc48203799)

[3.5.8 词性标注测试 22](#_Toc48203800)

[3.6 实验总结 24](#_Toc48203801)

[3.7 开放题（可选） 24](#_Toc48203802)

# 实验总览

## 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：

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；

## 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。

## 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 |

# CRF算法实现实验

## 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‾P (Y/X)；预测时，对于给定的输入序列x，求出条件概率‾P(y/x)最大的输出序列‾y。

【实验环境要求】:

1、python3.6

2、mindspore0.5

3、本机

## 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。

## 理论介绍

### 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：



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。

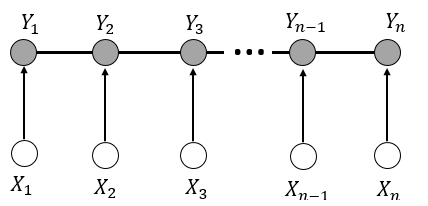


图2-1 线性链条件随机场

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：

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。

### 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：

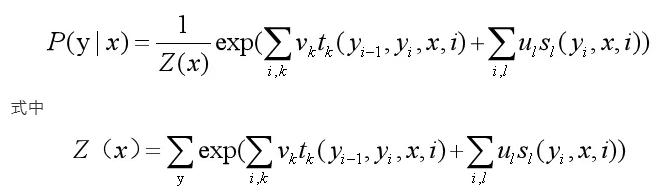


图2-3 线性链条件随机场的参数化表现形式

tk和sl是特征函数，vk和ul是对应的权值

tk是状态转移函数，vk是对应的权值；sl是发射函数，ul是对应的权值。假如所有的tk，sl 和vk，ul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。

## 代码实现

### 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

### CRF类的创建

class CRF(nn.Cell):

### 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.

'''

### 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()

### 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

### 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

### 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

### 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

### 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

### 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

# CRF实现词性标注实验

## 实验简介

本实验将展示CRF在词性标注方面的应用。

## 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。

## 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。

## 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。

## 实验步骤

### 安装包

1）安装keras-preprocessing

!pip install keras-preprocessing==1.0.9

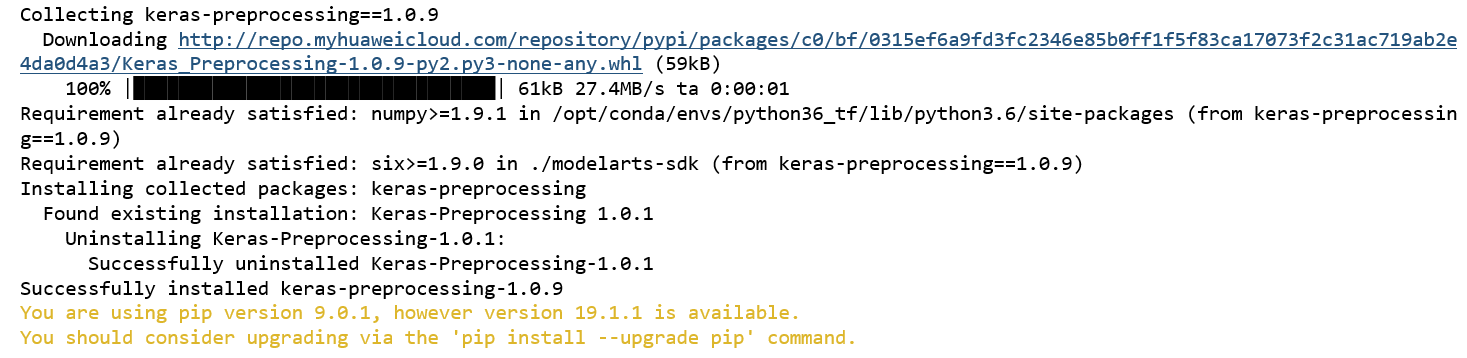


图2-4 安装keras-preprocessing

2）解压keras-contrib-master.zip

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

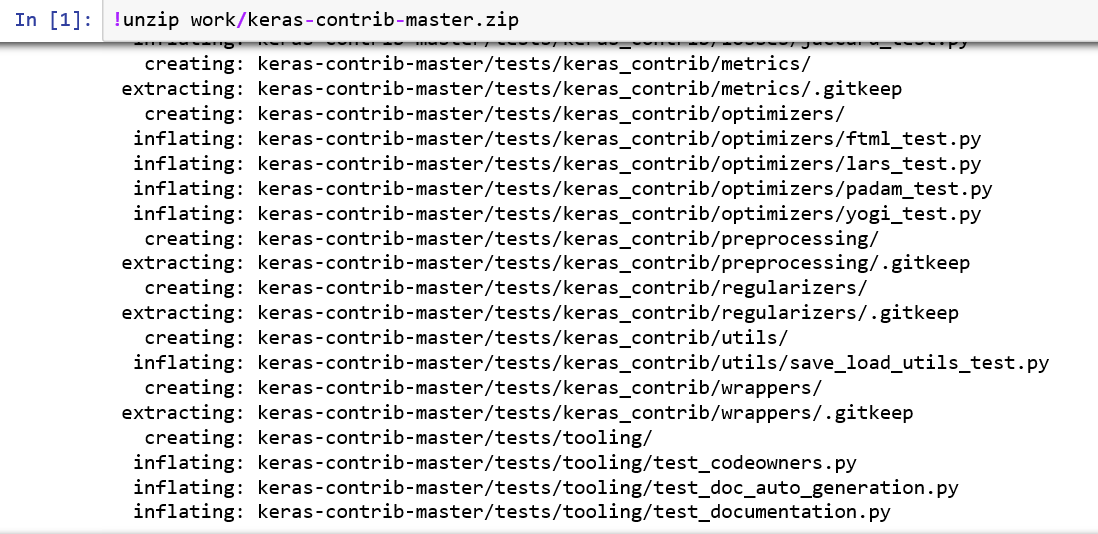


图2-5 解压keras-contrib-master.zip

3）安装keras-contrib-master

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

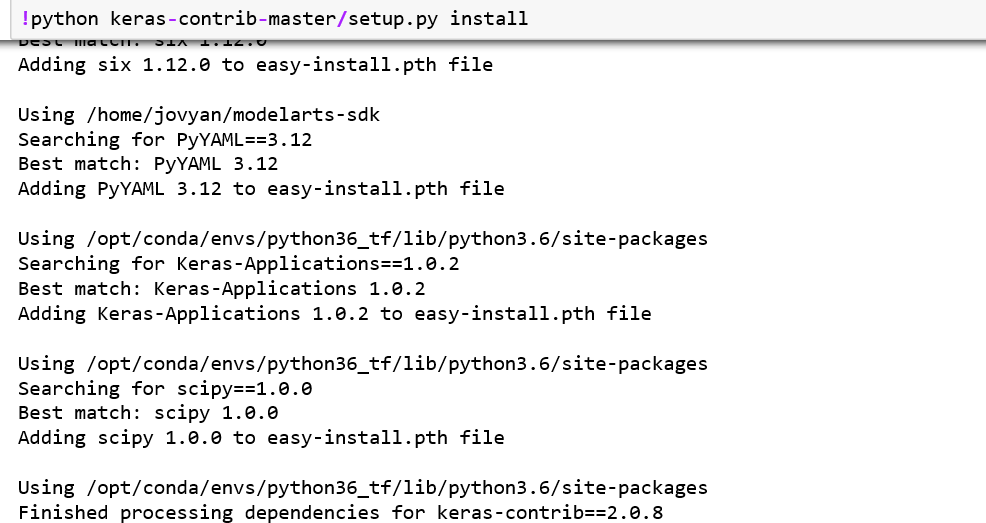


图2-6 安装keras-contrib-master

4）解压安装包bi-lstm-crf-master.zip

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

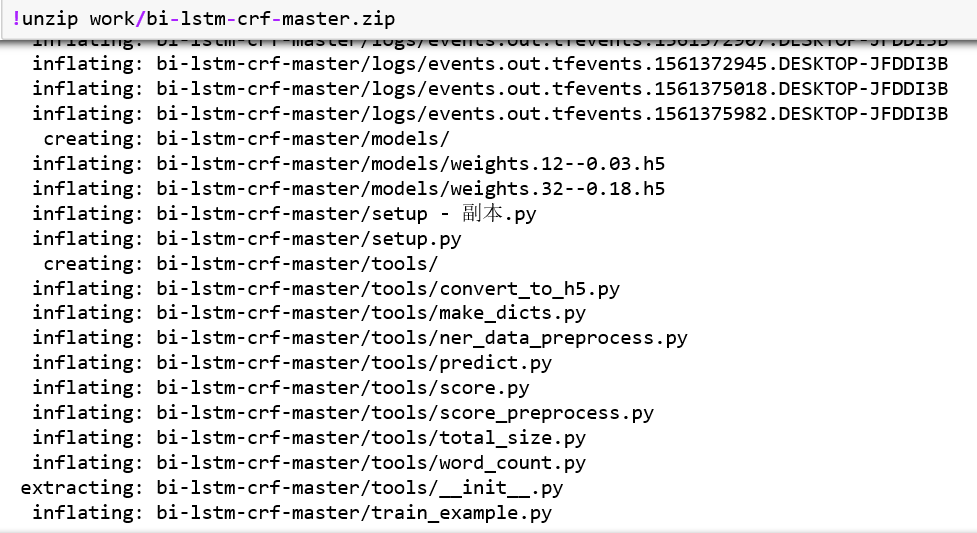


图2-7 解压bi-lstm-crf-master.zip

5）安装bi-lstm-crf

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

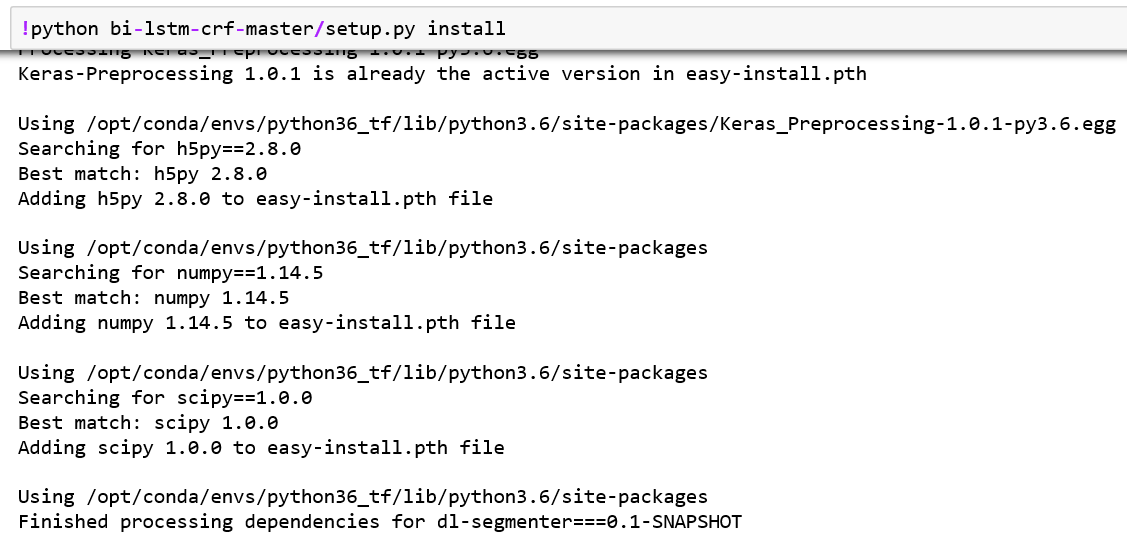


图2-8 安装bi-lstm-crf

### 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。

### 解压数据

!unzip work/people-2014.zip

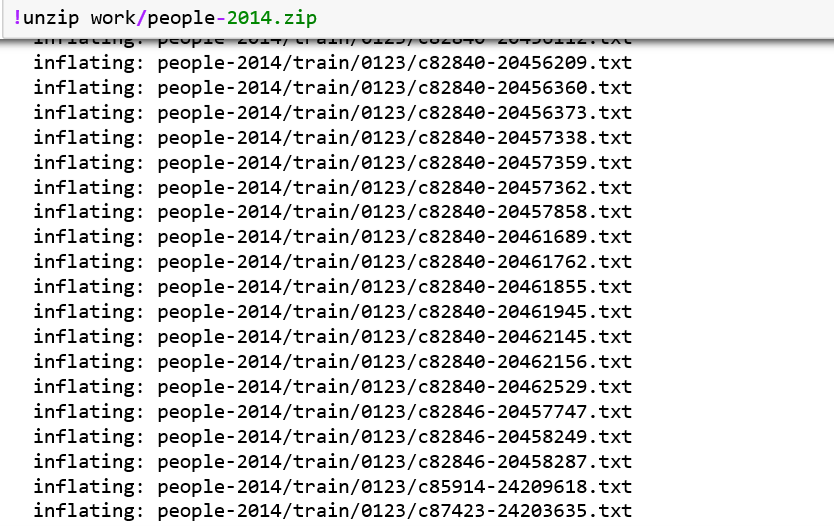


图2-9 解压需要使用的数据

### 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：

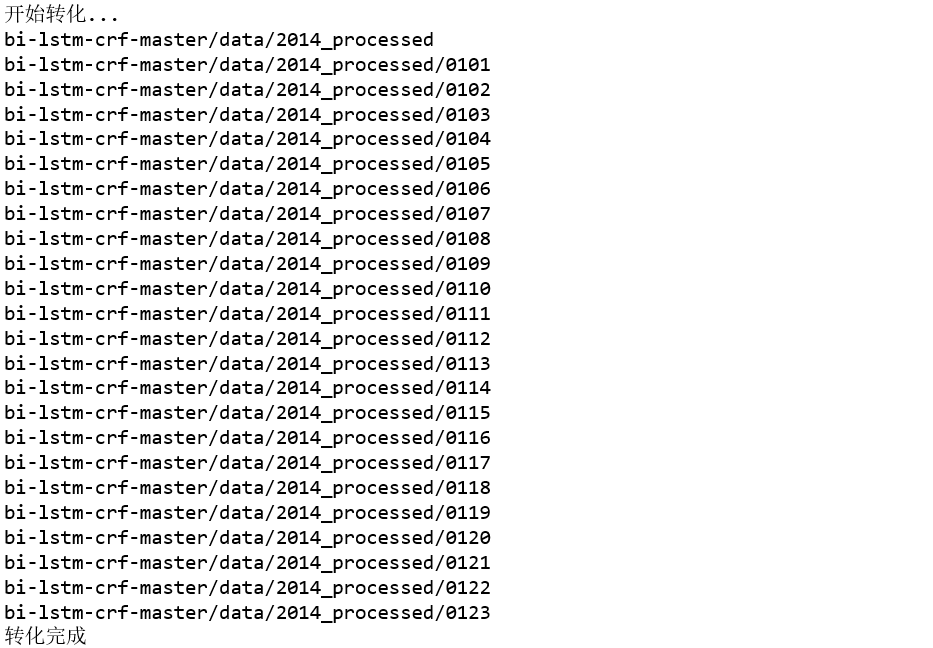


图2-10 语料转化过程

1. 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：



图2-11 生成字典过程

1. 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：

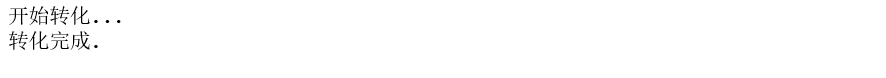


图2-12 转化h5文件过程

### 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

}

### 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)

### 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：

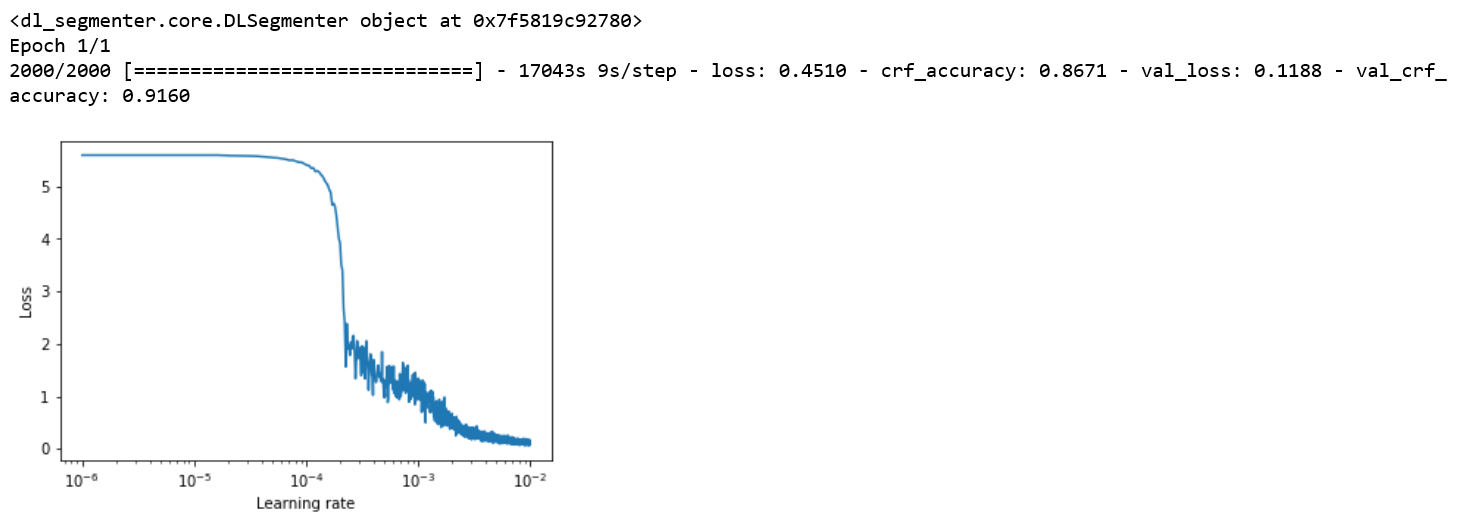


图2-13 完成训练后的结果

### 词性标注测试

1. 重启kernel

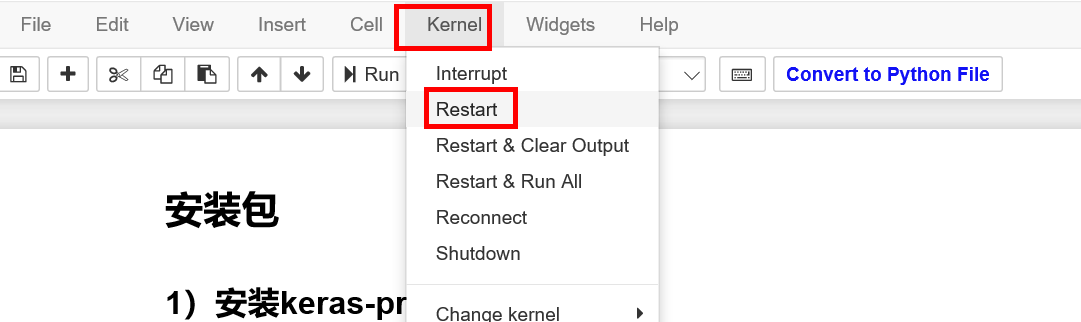


图2-14 重启kernel

1. 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

1. 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：

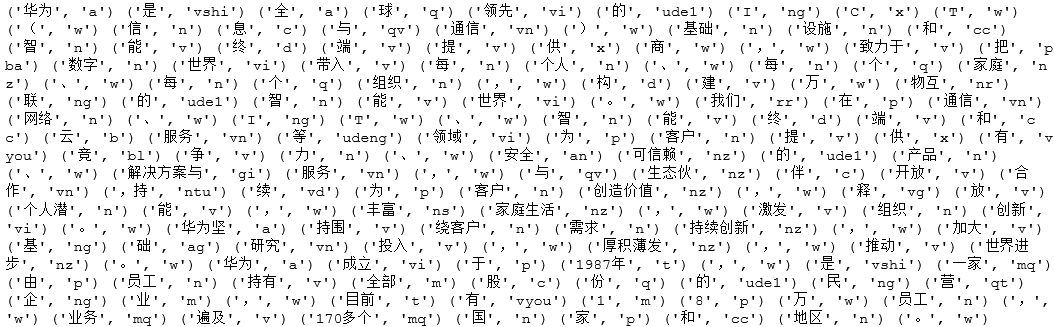


图3-12 词性标注的结果

## 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。

## 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？