

**课 程 实 验 报 告**

**课程名称： 自然语言处理实验**

**专业班级： 计算机科学与技术2005**

**学 号： U202015439**

**姓 名： 申智**

**指导教师： 姚德中**

**报告日期： 2023年 6 月 13 日**

**计算机科学与技术学院**

目 录

[中文分词实现 2](#_Toc137666407)

[1 问题描述 2](#_Toc137666408)

[1.1 实验目的 2](#_Toc137666409)

[1.2 实验任务 2](#_Toc137666410)

[1.2.1 基础任务 2](#_Toc137666411)

[1.3.2 选做任务 3](#_Toc137666412)

[2 模块实现 4](#_Toc137666413)

[2.1 基于词典 4](#_Toc137666414)

[2.1.1 前向最大匹配 4](#_Toc137666415)

[2.1.2 后向最大匹配 4](#_Toc137666416)

[2.1.3 双向最大匹配 4](#_Toc137666417)

[2.2 基于Bi-LSTM+CRF 5](#_Toc137666418)

[2.2.1 data\_u.py 5](#_Toc137666419)

[2.2.2 dataloader.py 5](#_Toc137666420)

[2.2.3 model.py 5](#_Toc137666421)

[2.2.4 run.py 5](#_Toc137666422)

[2.2.5 infer.py 5](#_Toc137666423)

[3 实验小结 7](#_Toc137666424)

[参考文献 8](#_Toc137666425)

[附录A 基于词典的源程序 9](#_Toc137666426)

[附录B 基于Bi-LSTM+CRF的源程序 13](#_Toc137666427)

# 中文分词实现

## 1 问题描述

中文分词指将汉字序列切分成单个词语的过程。在以英文为代表的拉丁语系中，单词之间是以空格作为自然分界符的。而中文只有字、句和段能通过明显的分界符来划界，唯独词没有一个形式上的分界符，这给中文的处理带来了独有的困难，因此中文分词技术得到了广泛研究。

理论上讲，构建一套完备的分词规则便可以将所有句子正确划分，但语言规则庞大复杂并且是动态发展的，编写这样一套规则是不现实的。

### 1.1 实验目的

通过实验了解中文分词的大致思路。在实验中实现中文分词处理，可考虑使用课堂讲解过的算法（比如基于统计、基于词典的分词方法等），或者课外学习算法（比如Bi-LSTM+CRF模型等）进行分词，最后对比不同算法分词效果和性能，加深对中文分词算法的理解。

### 1.2 实验任务

实验任务包括基础任务与选做任务。基础任务中需实现基于词典和基于统计的中文分词算法，完成后可获得实验课程基础分。选做任务中需要对基础任务中的分词器进行优化，选做部分的分数通过分词器在测试集上的表现决定。最终提交的实验报告中应包括基础任务完成情况与选做任务中采取的优化措施。

#### 1.2.1 基础任务

1. 实现基于词典的分词算法

实验一资料包下的“Dictionary\_based”文件夹中提供了基础词典和分词算法的大致框架。分词算法的核心部分需要大家完成，实验中提供了若干测试样本用以帮助大家判断算法是否正确实现。

1. 实现基于统计的分词算法

实验中给出Bi-LSTM+CRF模型的基础实现，相关代码及说明文档位于实验一资料包下的“Bi-LSTM+CRF”文件夹下。请根据给定的实验资料中README.md文件配置相应实验环境，说明：（1）提供源码PyTorch语言编写（可根据个人掌情况用其他语言编写），默认运行版本是CPU版本；（2）如希望运行NPU版本，大家可跟任课老师联系，申请华为云资源运行（需提前统计名单：姓名+学号+个人手机号码+邮箱）；

#### 1.3.2 选做任务

优化基础任务中实现的分词器，可考虑的优化方案有：

1. 修改网络结构，例如引入BERT等预训练语言模型；
2. 与命名实体识别算法相互配合，减少对命名实体的错误分割；
3. 构造合适的词典集（可扩充+人工整理）；
4. 实现新词发现（登录）功能，识别测试集中的新词（未登录词）；
5. 调整、优化模型训练过程中的超参数。

完成优化后对测试文件“Bi-LSTM+CRF/data/test.txt”进行分词，分词结果保存到.txt文件中utf-8编码，词与词之间以空格分隔，每个测试样本占一行。文件“Bi-LSTM+CRF/cws\_result.txt”中给出了输出示例。提交分词结果后，依据单词级别的F1-score进行评判，决定选做部分的实验分数。

单词级别的F1-score的计算方式如下：

Gold: 共同 创造 美好 的 新 世纪 —— 二○○一年 新年 贺词

Hypothesis: 共同 创造 美 好 的 新 世纪 —— 二○○一年 新年 贺词

Precision = 9 / 11 = 0.818

Recall = 9 / 10 = 0.9

F1-score = 2\*Precision\*Recall/(Precision+Recall)=0.857

## 2 模块实现

### 2.1 基于词典

Tokenizer类是一个分词器，其构造函数需要传入一个词典（words）、最大词长（max\_len）以及一个分隔符（delimiter）。其中，词典用于匹配待分词文本中的词语，最大词长用于限制分词算法的最大匹配长度，分隔符则用于对文本进行切分。

load\_dict(path)函数用于加载词典文件，并将所有词语存入一个set中，以便在分词时进行匹配。

#### 2.1.1 前向最大匹配

forward\_mm\_split()方法实现了前向最大匹配算法，其输入为待分词字符串，输出为分词结果，以list形式存放，每个元素为分出的词。该方法使用了一个while循环，不断将待分词字符串从左到右扫描，并取出前面的最长词语作为匹配结果。如果匹配成功，则将该词语加入分词结果列表中，并将已经匹配过的部分从待分词字符串中去掉，继续对剩余部分进行匹配。如果匹配失败，则将匹配长度减1，并重新匹配。

#### 2.1.2 后向最大匹配

reverse\_mm\_split()方法实现了后向最大匹配算法，与前向最大匹配算法类似，首先设定一个固定长度的窗口（即 word\_len），从字符串的右端开始扫描，每次取出一个子串，然后在字典 words 中查找是否存在这个子串作为词语。如果存在，则将其作为一个词加入到结果列表 word\_list 中，并将扫描位置向左移动 word\_len 位，然后将 word\_len 重新设置为固定长度；否则将 word\_len 减1，继续进行查找和匹配的过程。直到整个字符串扫描完成，返回结果列表 word\_list。

#### 2.1.3 双向最大匹配

bidirectional\_mm\_split()方法实现了双向最大匹配算法，先采用 FMM 算法得到一组分词结果 forward，再采用 RMM 算法得到另一组分词结果 reverse。然后比较两种分词结果的差异性，统计出每种分词结果的总词数（total\_words）、单字词个数（single\_words）和非字典词数（illegal\_words），并根据这些统计信息计算出罚分（score）。最后选择罚分值最小的一组结果作为最终的分词结果并返回。

### 2.2 基于Bi-LSTM+CRF

#### 2.2.1 data\_u.py

处理中文分词数据并将处理后的数据保存在pickle文件中。

1. getList接受单个单词并基于单词长度返回标签序列。
2. handle\_data逐行读取输入文件，将单词转换为ID和标签，并将它们分别存储在x\_data和y\_data中。然后，它使用sklearn中的train\_test\_split方法将数据拆分为训练集和测试集。最后，它将所有必要的变量保存到pickle文件中以便以后使用。

#### 2.2.2 dataloader.py

读取通过data\_u.py处理完后的文件data\_save.pkl，并将其向量化。

#### 2.2.3 model.py

定义了一个名为 CWS 的 PyTorch 模型类，实现中文分词任务。

1. \_\_init\_\_ 方法初始化了模型的一些基本参数，例如词汇表大小、嵌入维度和隐藏层维度。它还初始化了嵌入层、LSTM 和 CRF 层。
2. init\_hidden 方法用于初始化 LSTM 隐藏状态。
3. \_get\_lstm\_features 方法将输入的句子通过嵌入层转换为嵌入向量，然后使用双向 LSTM 进行特征提取。该方法还使用线性层将 LSTM 输出映射到标签空间，并返回CRF的输入特征。
4. forward 方法计算损失并返回。它首先调用 \_get\_lstm\_features 获取CRF的输入特征，然后使用 CRF 层计算损失。
5. infer 方法执行预测并返回结果。它也首先调用 \_get\_lstm\_features 获取CRF的输入特征，然后使用 CRF 层进行解码，得到最终的预测结果。

#### 2.2.4 run.py

采用小批量梯度下降法，对模型进行训练，使得loss值降低。

小批量梯度下降，是对批量梯度下降以及随机梯度下降的一个折中办法。其思想是：每次迭代 使用 batch\_size个样本来对参数进行更新，每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。

#### 2.2.5 infer.py

使用之前训练好的CWS模型对测试数据进行分词，并将结果输出到 'cws\_result.txt' 文件。遍历每一个测试样本并对其进行预测。对于样本中的每个字符，将字符转换为该字符在词汇表中的索引（如果存在），否则将其映射到未知字符的索引。接下来，调用模型的infer方法来进行预测，得到一个标签序列。最后将输入文本和预测的标签序列一起输出到文件 'cws\_result.txt' 中，用空格表示分词位置。

## 3 实验小结

基于词典：实验代码的总体框架由老师给出，仅需补充前向最大匹配、后向最大匹配和双向最大匹配三个函数的具体实现。根据三个函数的规则可以轻松写出相应代码。预测结果与预期相符。

基于Bi-LSTM+CRF：相关代码已由老师给出，配置好环境即可运行。只是尝试性地调整了一下超参数和增加部分训练集，以期提高模型的精度。

总体来说，这次中文分词实验的工作量不大，但涉及到两种不同类型的分词模型，有助于理解中文分词的基本原理和应用场景，并且帮助我学习了如何使用相关的开源工具和技术。

# 参考文献

[1] 郑捷著. NLP汉语自然语言处理---原理与实践. 电子工业出版社

# 附录A 基于词典的源程序

class Tokenizer(object):

def \_\_init\_\_(self, init\_words, init\_max\_len, init\_delimiter):

self.words = init\_words

self.max\_len = init\_max\_len

self.delimiter = init\_delimiter

def forward\_mm\_split(self, fmm\_text):

"""

正向最大匹配分词算法

:param fmm\_text: 待分词字符串

:return: 分词结果，以list形式存放，每个元素为分出的词

"""

# 字词列表，存放分词结果

word\_list = []

# 用于记录分词的起始位置

count = 0

# 字或词当前的长度

word\_len = self.max\_len

while word\_len > 0 and count < len(fmm\_text):

word = fmm\_text[count:count + word\_len]

word\_len = len(word)

if (word in self.words) or (word in self.delimiter):

word\_list.append(word)

count = count + word\_len

word\_len = self.max\_len

else:

word\_len = word\_len - 1

return word\_list

def reverse\_mm\_split(self, rmm\_text):

"""

逆向最大匹配分词算法

:param rmm\_text: 待分词字符串

:return: 分词结果，以list形式存放，每个元素为分出的词

"""

# 字词列表，存放分词结果

word\_list = []

# 用于记录分词的末尾位置

count = len(rmm\_text)

# 字或词当前的长度

word\_len = self.max\_len

while word\_len > 0 and count > 0:

if count <= word\_len:

word = rmm\_text[:count]

else:

word = rmm\_text[(count - word\_len):count]

word\_len = len(word)

if (word in self.words) or (word in self.delimiter):

word\_list.insert(0, word)

count = count - word\_len

word\_len = self.max\_len

else:

word\_len = word\_len - 1

return word\_list

def bidirectional\_mm\_split(self, bi\_text):

"""

双向最大匹配分词算法

:param bi\_text: 待分词字符串

:return: 分词结果，以list形式存放，每个元素为分出的词

"""

# 前向最大匹配得到的分词结果

forward = self.forward\_mm\_split(bi\_text)

# 后向最大匹配得到的分词结果

reverse = self.reverse\_mm\_split(bi\_text)

# 总词数

forward\_total\_words = len(forward)

reverse\_total\_words = len(reverse)

# 单字词个数

forward\_single\_words = 0

reverse\_single\_words = 0

# 非字典词数

forward\_illegal\_words = 0

reverse\_illegal\_words = 0

# 罚分，分值越低，表明结果越好

forward\_score = 0

reverse\_score = 0

if forward == reverse:

return forward

else:

# 统计前向匹配的各个词情况

for word in forward:

if len(word) == 1:

forward\_single\_words += 1

if word not in self.words:

forward\_illegal\_words += 1

# 统计后向匹配的各个词情况

for word in reverse:

if len(word) == 1:

reverse\_single\_words += 1

if word not in self.words:

reverse\_illegal\_words += 1

# 计算罚分

if forward\_total\_words < reverse\_total\_words:

reverse\_score += 1

else:

forward\_score += 1

if forward\_illegal\_words < reverse\_illegal\_words:

reverse\_score += 1

else:

forward\_score += 1

if forward\_single\_words < reverse\_single\_words:

reverse\_score += 1

else:

forward\_score += 1

# 比较罚分情况，罚分最小的选做最终结果

if forward\_score < reverse\_score:

return forward

else:

return reverse

def load\_dict(path):

tmp = set()

with open(path, 'r', encoding='utf-8') as f:

for line in f:

word = line.strip().split(' ')[0]

tmp.add(word)

return tmp

if \_\_name\_\_ == '\_\_main\_\_':

words = load\_dict('dict.txt')

max\_len = max(map(len, [word for word in words]))

delimiter = ',./;\'<>?:\"-=\_+!@#$%^&\*(){}，。？、；‘“’”·~[]'

# test

tokenizer = Tokenizer(words, max\_len, delimiter)

texts = [

'研究生命的起源',

'无线电法国别研究',

'人要是行，干一行行一行，一行行行行行，行行行干哪行都行。'

]

for text in texts:

# 前向最大匹配

print('前向最大匹配:', '/'.join(tokenizer.forward\_mm\_split(text)))

# 后向最大匹配

print('后向最大匹配:', '/'.join(tokenizer.reverse\_mm\_split(text)))

# 双向最大匹配

print('双向最大匹配:', '/'.join(tokenizer.bidirectional\_mm\_split(text)))

print('')

# 附录B 基于Bi-LSTM+CRF的源程序

1. **data\_u.py**

import codecs

from sklearn.model\_selection import train\_test\_split

import pickle

INPUT\_DATA = "train.txt"

SAVE\_PATH = "./datasave.pkl"

id2tag = ['B', 'M', 'E', 'S'] # B：分词头部 M：分词词中 E：分词词尾 S：独立成词

tag2id = {'B': 0, 'M': 1, 'E': 2, 'S': 3}

word2id = {}

id2word = []

def getList(input\_str):

'''

单个分词转换为tag序列

:param input\_str: 单个分词

:return: tag序列

'''

outpout\_str = []

if len(input\_str) == 1:

outpout\_str.append(tag2id['S'])

elif len(input\_str) == 2:

outpout\_str = [tag2id['B'], tag2id['E']]

else:

M\_num = len(input\_str) - 2

M\_list = [tag2id['M']] \* M\_num

outpout\_str.append(tag2id['B'])

outpout\_str.extend(M\_list)

outpout\_str.append(tag2id['E'])

return outpout\_str

def handle\_data():

'''

处理数据，并保存至savepath

:return:

'''

x\_data = []

y\_data = []

wordnum = 0

line\_num = 0

with open(INPUT\_DATA, 'r', encoding="utf-8") as ifp:

for line in ifp:

line\_num = line\_num + 1

line = line.strip()

if not line:

continue

line\_x = []

for i in range(len(line)):

if line[i] == " ":

continue

if (line[i] in id2word):

line\_x.append(word2id[line[i]])

else:

id2word.append(line[i])

word2id[line[i]] = wordnum

line\_x.append(wordnum)

wordnum = wordnum + 1

x\_data.append(line\_x)

lineArr = line.split()

line\_y = []

for item in lineArr:

line\_y.extend(getList(item))

y\_data.append(line\_y)

print(x\_data[0])

print([id2word[i] for i in x\_data[0]])

print(y\_data[0])

print([id2tag[i] for i in y\_data[0]])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size=0.1, random\_state=43)

with open(SAVE\_PATH, 'wb') as outp:

pickle.dump(word2id, outp)

pickle.dump(id2word, outp)

pickle.dump(tag2id, outp)

pickle.dump(id2tag, outp)

pickle.dump(x\_train, outp)

pickle.dump(y\_train, outp)

pickle.dump(x\_test, outp)

pickle.dump(y\_test, outp)

if \_\_name\_\_ == "\_\_main\_\_":

handle\_data()

1. **dataloader.py**

import torch

import pickle

from torch.utils.data import Dataset, DataLoader

from torch.nn.utils.rnn import pad\_sequence

class Sentence(Dataset):

def \_\_init\_\_(self, x, y, batch\_size=10):

self.x = x

self.y = y

self.batch\_size = batch\_size

def \_\_len\_\_(self):

return len(self.x)

def \_\_getitem\_\_(self, idx):

assert len(self.x[idx]) == len(self.y[idx])

return self.x[idx], self.y[idx]

@staticmethod

def collate\_fn(train\_data):

train\_data.sort(key=lambda data: len(data[0]), reverse=True)

data\_length = [len(data[0]) for data in train\_data]

data\_x = [torch.LongTensor(data[0]) for data in train\_data]

data\_y = [torch.LongTensor(data[1]) for data in train\_data]

mask = [torch.ones(l, dtype=torch.uint8) for l in data\_length]

data\_x = pad\_sequence(data\_x, batch\_first=True, padding\_value=0)

data\_y = pad\_sequence(data\_y, batch\_first=True, padding\_value=0)

mask = pad\_sequence(mask, batch\_first=True, padding\_value=0)

return data\_x, data\_y, mask, data\_length

if \_\_name\_\_ == '\_\_main\_\_':

# test

with open('../data/datasave.pkl', 'rb') as inp:

word2id = pickle.load(inp)

id2word = pickle.load(inp)

tag2id = pickle.load(inp)

id2tag = pickle.load(inp)

x\_train = pickle.load(inp)

y\_train = pickle.load(inp)

x\_test = pickle.load(inp)

y\_test = pickle.load(inp)

train\_dataloader = DataLoader(Sentence(x\_train, y\_train), batch\_size=10, shuffle=True, collate\_fn=Sentence.collate\_fn)

for input, label, mask, length in train\_dataloader:

print(input, label)

break

1. **model.py**

import torch

import torch.nn as nn

from torchcrf import CRF

from torch.nn.utils.rnn import pack\_padded\_sequence, pad\_packed\_sequence

class CWS(nn.Module):

def \_\_init\_\_(self, vocab\_size, tag2id, embedding\_dim, hidden\_dim):

super(CWS, self).\_\_init\_\_()

self.embedding\_dim = embedding\_dim

self.hidden\_dim = hidden\_dim

self.vocab\_size = vocab\_size

self.tag2id = tag2id

self.tagset\_size = len(tag2id)

self.word\_embeds = nn.Embedding(vocab\_size + 1, embedding\_dim)

self.lstm = nn.LSTM(embedding\_dim, hidden\_dim // 2, num\_layers=1,

bidirectional=True, batch\_first=True)

self.hidden2tag = nn.Linear(hidden\_dim, self.tagset\_size)

self.crf = CRF(4, batch\_first=True)

def init\_hidden(self, batch\_size, device):

return (torch.randn(2, batch\_size, self.hidden\_dim // 2, device=device),

torch.randn(2, batch\_size, self.hidden\_dim // 2, device=device))

def \_get\_lstm\_features(self, sentence, length):

batch\_size, seq\_len = sentence.size(0), sentence.size(1)

# idx->embedding

embeds = self.word\_embeds(sentence.view(-1)).reshape(batch\_size, seq\_len, -1)

embeds = pack\_padded\_sequence(embeds, length, batch\_first=True)

# LSTM forward

self.hidden = self.init\_hidden(batch\_size, sentence.device)

lstm\_out, self.hidden = self.lstm(embeds, self.hidden)

lstm\_out, \_ = pad\_packed\_sequence(lstm\_out, batch\_first=True)

lstm\_feats = self.hidden2tag(lstm\_out)

return lstm\_feats

def forward(self, sentence, tags, mask, length):

emissions = self.\_get\_lstm\_features(sentence, length)

loss = -self.crf(emissions, tags, mask, reduction='mean')

return loss

def infer(self, sentence, mask, length):

emissions = self.\_get\_lstm\_features(sentence, length)

return self.crf.decode(emissions, mask)

1. **run.py**

import pickle

import logging

import argparse

import os

import torch

from torch.utils.data import DataLoader

from torch.optim import Adam

from model import CWS

from dataloader import Sentence

def get\_param():

parser = argparse.ArgumentParser()

parser.add\_argument('--embedding\_dim', type=int, default=100)

parser.add\_argument('--lr', type=float, default=0.005)

parser.add\_argument('--max\_epoch', type=int, default=10)

parser.add\_argument('--batch\_size', type=int, default=128)

parser.add\_argument('--hidden\_dim', type=int, default=200)

parser.add\_argument('--cuda', action='store\_true', default=False)

return parser.parse\_args()

def set\_logger():

log\_file = os.path.join('save', 'log.txt')

logging.basicConfig(

format='%(asctime)s %(levelname)-8s %(message)s',

level=logging.DEBUG,

datefmt='%Y-%m%d %H:%M:%S',

filename=log\_file,

filemode='w',

)

console = logging.StreamHandler()

console.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s %(levelname)-8s %(message)s')

console.setFormatter(formatter)

logging.getLogger('').addHandler(console)

def entity\_split(x, y, id2tag, entities, cur):

start, end = -1, -1

for j in range(len(x)):

if id2tag[y[j]] == 'B':

start = cur + j

elif id2tag[y[j]] == 'M' and start != -1:

continue

elif id2tag[y[j]] == 'E' and start != -1:

end = cur + j

entities.add((start, end))

start, end = -1, -1

elif id2tag[y[j]] == 'S':

entities.add((cur + j, cur + j))

start, end = -1, -1

else:

start, end = -1, -1

def main(args):

use\_cuda = args.cuda and torch.cuda.is\_available()

with open('data/datasave.pkl', 'rb') as inp:

word2id = pickle.load(inp)

id2word = pickle.load(inp)

tag2id = pickle.load(inp)

id2tag = pickle.load(inp)

x\_train = pickle.load(inp)

y\_train = pickle.load(inp)

x\_test = pickle.load(inp)

y\_test = pickle.load(inp)

model = CWS(len(word2id), tag2id, args.embedding\_dim, args.hidden\_dim)

if use\_cuda:

model = model.cuda()

for name, param in model.named\_parameters():

logging.debug('%s: %s, require\_grad=%s' % (name, str(param.shape), str(param.requires\_grad)))

optimizer = Adam(model.parameters(), lr=args.lr)

train\_data = DataLoader(

dataset=Sentence(x\_train, y\_train),

shuffle=True,

batch\_size=args.batch\_size,

collate\_fn=Sentence.collate\_fn,

drop\_last=False,

num\_workers=6

)

test\_data = DataLoader(

dataset=Sentence(x\_test[:1000], y\_test[:1000]),

shuffle=False,

batch\_size=args.batch\_size,

collate\_fn=Sentence.collate\_fn,

drop\_last=False,

num\_workers=6

)

for epoch in range(args.max\_epoch):

step = 0

log = []

for sentence, label, mask, length in train\_data:

if use\_cuda:

sentence = sentence.cuda()

label = label.cuda()

mask = mask.cuda()

# forward

loss = model(sentence, label, mask, length)

log.append(loss.item())

# backward

optimizer.zero\_grad()

loss.backward()

optimizer.step()

step += 1

if step % 100 == 0:

logging.debug('epoch %d-step %d loss: %f' % (epoch, step, sum(log)/len(log)))

log = []

# test

entity\_predict = set()

entity\_label = set()

with torch.no\_grad():

model.eval()

cur = 0

for sentence, label, mask, length in test\_data:

if use\_cuda:

sentence = sentence.cuda()

label = label.cuda()

mask = mask.cuda()

predict = model.infer(sentence, mask, length)

for i in range(len(length)):

entity\_split(sentence[i, :length[i]], predict[i], id2tag, entity\_predict, cur)

entity\_split(sentence[i, :length[i]], label[i, :length[i]], id2tag, entity\_label, cur)

cur += length[i]

right\_predict = [i for i in entity\_predict if i in entity\_label]

if len(right\_predict) != 0:

precision = float(len(right\_predict)) / len(entity\_predict)

recall = float(len(right\_predict)) / len(entity\_label)

logging.info("precision: %f" % precision)

logging.info("recall: %f" % recall)

logging.info("fscore: %f" % ((2 \* precision \* recall) / (precision + recall)))

else:

logging.info("precision: 0")

logging.info("recall: 0")

logging.info("fscore: 0")

model.train()

path\_name = "./save/model\_epoch" + str(epoch) + ".pkl"

torch.save(model, path\_name)

logging.info("model has been saved in %s" % path\_name)

if \_\_name\_\_ == '\_\_main\_\_':

set\_logger()

main(get\_param())

1. **infer.py**

import torch

import pickle

if \_\_name\_\_ == '\_\_main\_\_':

model = torch.load('save/model.pkl', map\_location=torch.device('cpu'))

output = open('cws\_result.txt', 'w', encoding='utf-8')

with open('data/datasave.pkl', 'rb') as inp:

word2id = pickle.load(inp)

id2word = pickle.load(inp)

tag2id = pickle.load(inp)

id2tag = pickle.load(inp)

x\_train = pickle.load(inp)

y\_train = pickle.load(inp)

x\_test = pickle.load(inp)

y\_test = pickle.load(inp)

with open('data/test.txt', 'r', encoding='utf-8') as f:

for test in f:

flag = False

test = test.strip()

x = torch.LongTensor(1, len(test))

mask = torch.ones\_like(x, dtype=torch.uint8)

length = [len(test)]

for i in range(len(test)):

if test[i] in word2id:

x[0, i] = word2id[test[i]]

else:

x[0, i] = len(word2id)

predict = model.infer(x, mask, length)[0]

for i in range(len(test)):

print(test[i], end='', file=output)

if id2tag[predict[i]] in ['E', 'S']:

print(' ', end='', file=output)

print(file=output)