获取网站中的内容

获取网址；URLLIB爬虫

获取网站中的内容

# encoding=utf-8

import jieba

a=["C:\\Users\amin\Desktop\jianggee.txt"]

print(a)

'''

f=open(r"C:\\Users\amin\Desktop\jianggee.txt",'r')

s=f.read()

print (s)

seg\_list = jieba.cut(r, cut\_all=False)

print("Default Mode: " + "/ ".join(seg\_list))  # 精确模式

import urllib.request

response = urllib.request.urlopen("http://blog.csdn.net/kangqianglong/article/details/53173889")

b = response.read()

b = str(b, encoding='utf-8')

print (b)

'''

#coding=utf-8

import urllib

from bs4 import BeautifulSoup

import sys

import re

import urllib.request

import urllib.parse

with open('data.txt','w') as f:

    key\_word = []

    with open('key\_word.txt','r') as kf:

        for line in kf:

            request = urllib.request.Request('http://www.baidu.com/s?wd='+urllib.parse.quote(str(line.strip()).encode('gbk')))#.decode(sys.stdin.encoding)

            response = urllib.request.urlopen(request)

            soup = BeautifulSoup(response.read())

            data = [re.sub(u'<[\d\D]\*?>',' ',str(item)) for item in soup.select('div.result h3.t > a')]

            for item in data:

                f.writelines(''.join(item.strip().split())+'\n')

#!/usr/bin/python

#coding=utf-8

import re

import requests

from pyquery import PyQuery as Pq

class BaiduSearchSpider(object):

    def \_\_init\_\_(self, searchText):

        self.url = "http://www.baidu.com/baidu?wd=%s&tn=monline\_4\_dg" % searchText

        self.headers = {"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_3) AppleWebKit/600.5.17 (KHTML, like Gecko) Version/8.0.5 Safari/600.5.17"}

        self.\_page = None

    @property

    def page(self):

        if not self.\_page:

            r = requests.get(self.url, headers=self.headers)

            r.encoding = 'utf-8'

            self.\_page = Pq(r.text)

        return self.\_page

    @property

    def baiduURLs(self):

        return [(site.attr('href'), site.text().encode('utf-8')) for site in self.page('div.result.c-container  h3.t  a').items()]

    @property

    def originalURLs(self):

        tmpURLs = self.baiduURLs

        print (tmpURLs)

        originalURLs = []

        for tmpurl in tmpURLs:

            tmpPage = requests.get(tmpurl[0])

            #tmpPage.encoding = 'utf-8' #这样不好使，print的时候python报错

            tmptext = tmpPage.text.encode('utf-8')

            urlMatch = re.search(r'URL=\'(.\*?)\'', tmptext, re.S)

            if not urlMatch == None:

                print (urlMatch.group(1), "   ", tmpurl[1])

                originalURLs.append(tmpurl)

            else:

                print ("---------------")

                print ("No Original URL found!!")

                print (tmpurl[0])

                print (tmpurl[1])

        return originalURLs

searchText = input("搜索内容是：")

print (searchText)

bdsearch = BaiduSearchSpider(searchText)

originalurls = bdsearch.originalURLs

print ('=======Original URLs========')

print (originalurls)

print ('============================')

判别学习

判别学习；网址摘录

str="<!DOCTYPE html><!--STATUS OK--><html><head><meta charset="UTF-8"><meta http-equiv="X-UA-Compatible" content="IE=Edge" /><meta name="referrer" content="always" /><meta name="description" content="陈世峰，男，1991年1月19日出生于宁夏吴忠盐池县，籍贯陕西榆林定边县，华侨大学华文学院2013届毕业生，日本东京大东文化大学汉语研究科研究生。2016年11月3日凌晨，在日本东京中野区一栋公寓门外，24岁中国女留学生江歌被其同住室友刘鑫的前男友陈世峰捅了数刀后身亡。2017年11月，一段江歌母亲..."><title>陈世峰（江歌案犯罪嫌疑人）\_百度百科</title><link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" /><link rel="icon" sizes="any" mask href="//www.baidu.com/img/baidu.svg"><meta name="keywords" content="陈世峰 陈世峰人物经历 陈世峰家庭生活 陈世峰人物事件"><meta name="image" content="https://bkssl.bdimg.com/cms/static/baike.png"><script type="text/javascript">  // 配置 PD 监控。  window.alogObjectConfig = {    product: '103',    page: '103\_1',    speed: {      sample: '0.008'    },    monkey: {      sample: '1',      hid: '1533'    },    exception: {       sample: '0.004'    },    feature: {      sample: '0.004'    },    csp: {      sample: '0.008',      'default-src': [        {match: '\*.baidu.com,\*.bdimg.com,localhost', target: 'Accept'},        {match: '\*', target: 'Accept,Warn'}      ]    }  };  void function(a,b,c,d,e,f,g){a.alogObjectName=e,a[e]=a[e]||function(){(a[e].q=a[e].q||[]).push(arguments)},a[e].l=a[e].l||+new Date,d="https:"===a.location.protocol?"https://fex.bdstatic.com"+d:"http://fex.bdstatic.com"+d;var h=!0;if(a.alogObjectConfig&&a.alogObjectConfig.sample){var i=Math.random();a.alogObjectConfig.rand=i,i>a.alogObjectConfig.sample&&(h=!1)}h&&(f=b.createElement(c),f.async=!0,f.src=d+"?v="+~(new Date/864e5)+~(new Date/864e5),g=b.getElementsByTagName(c)[0],g.parentNode.insertBefore(f,g))}(window,document,"script","/hunter/alog/alog.min.js","alog"),void function(){function a(){}window.PDC={mark:function(a,b){alog("speed.set",a,b||+new Date),alog.fire&&alog.fire("mark")},init:function(a){alog("speed.set","options",a)},view\_start:a,tti:a,page\_ready:a}}();  void function(n){var o=!1;n.onerror=function(n,e,t,c){var i=!0;return!e&&/^script error/i.test(n)&&(o?i=!1:o=!0),i&

Tensorflow模型判别

Tensorflow 模型判别

#! /usr/bin/env python

import tensorflow as tf  
import numpy as np  
import os  
import time  
import datetime  
import data\_helpers  
from text\_cnn import TextCNN  
from tensorflow.contrib import learn  
import csv

# Parameters  
# ==================================================

# Data Parameters  
tf.flags.DEFINE\_string("positive\_data\_file", "./data/test\_text/pos.txt", "Data source for the positive data.")  
tf.flags.DEFINE\_string("negative\_data\_file", "./data/test\_text/neg.txt", "Data source for the negative data.")

# Eval Parameters  
tf.flags.DEFINE\_integer("batch\_size", 64, "Batch Size (default: 64)")  
tf.flags.DEFINE\_string("checkpoint\_dir", "", "Checkpoint directory from training run")  
tf.flags.DEFINE\_boolean("eval\_train", False, "Evaluate on all training data")

# Misc Parameters  
tf.flags.DEFINE\_boolean("allow\_soft\_placement", True, "Allow device soft device placement")  
tf.flags.DEFINE\_boolean("log\_device\_placement", False, "Log placement of ops on devices")

FLAGS = tf.flags.FLAGS  
FLAGS.\_parse\_flags()  
print("\nParameters:")  
for attr, value in sorted(FLAGS.\_\_flags.items()):  
    print("{}={}".format(attr.upper(), value))  
print("")

# CHANGE THIS: Load data. Load your own data here  
if FLAGS.eval\_train:  
    x\_raw, y\_test = data\_helpers.load\_test\_data\_and\_labels(FLAGS.positive\_data\_file, FLAGS.negative\_data\_file)  
    y\_test = np.argmax(y\_test, axis=1)  
else:  
    x\_raw = ["a masterpiece four years in the making", "everything is off."]  
    y\_test = [1, 0]

# Map data into vocabulary  
vocab\_path = os.path.join(FLAGS.checkpoint\_dir, "..", "vocab")  
vocab\_processor = learn.preprocessing.VocabularyProcessor.restore(vocab\_path)

text\_list=[]

for text in x\_raw:  
    text\_list.append(' '.join(text))

x\_test = np.array(list(vocab\_processor.transform(text\_list)))

print("\nEvaluating...\n")

# Evaluation  
# ==================================================  
checkpoint\_file = tf.train.latest\_checkpoint(FLAGS.checkpoint\_dir)  
graph = tf.Graph()  
with graph.as\_default():  
    session\_conf = tf.ConfigProto(  
      allow\_soft\_placement=FLAGS.allow\_soft\_placement,  
      log\_device\_placement=FLAGS.log\_device\_placement)  
    sess = tf.Session(config=session\_conf)  
    with sess.as\_default():  
        # Load the saved meta graph and restore variables  
        saver = tf.train.import\_meta\_graph("{}.meta".format(checkpoint\_file))  
        saver.restore(sess, checkpoint\_file)

        # Get the placeholders from the graph by name  
        input\_x = graph.get\_operation\_by\_name("input\_x").outputs[0]  
        # input\_y = graph.get\_operation\_by\_name("input\_y").outputs[0]  
        dropout\_keep\_prob = graph.get\_operation\_by\_name("dropout\_keep\_prob").outputs[0]

        # Tensors we want to evaluate  
        predictions = graph.get\_operation\_by\_name("output/predictions").outputs[0]

        # Generate batches for one epoch  
        batches = data\_helpers.batch\_iter(list(x\_test), FLAGS.batch\_size, 1)

        # Collect the predictions here  
        all\_predictions = []

        for x\_test\_batch in batches:  
            batch\_predictions = sess.run(predictions, {input\_x: x\_test\_batch, dropout\_keep\_prob: 1.0})  
            all\_predictions = np.concatenate([all\_predictions, batch\_predictions])

# Print accuracy if y\_test is defined  
if y\_test is not None:  
    correct\_predictions = float(sum(all\_predictions == y\_test))  
    print("Total number of test examples: {}".format(len(y\_test)))  
    print("Accuracy: {:g}".format(correct\_predictions/float(len(y\_test))))

# Save the evaluation to a csv  
predictions\_human\_readable = np.column\_stack((np.array(x\_raw), all\_predictions))  
out\_path = os.path.join(FLAGS.checkpoint\_dir, "..", "prediction.csv")  
print("Saving evaluation to {0}".format(out\_path))  
with open(out\_path, 'w') as f:  
    csv.writer(f).writerows(predictions\_human\_readable)

#! /usr/bin/env python

import tensorflow as tf  
import numpy as np  
import os  
import time  
import datetime  
import data\_helpers  
from text\_cnn import TextCNN  
from tensorflow.contrib import learn  
import jieba  
# Parameters  
# ==================================================

# Data loading params  
tf.flags.DEFINE\_float("dev\_sample\_percentage", .1, "Percentage of the training data to use for validation")  
tf.flags.DEFINE\_string("positive\_data\_file", "./data/chinese/pos.txt", "Data source for the positive data.")  
tf.flags.DEFINE\_string("negative\_data\_file", "./data/chinese/neg.txt", "Data source for the negative data.")

# Model Hyperparameters  
tf.flags.DEFINE\_integer("embedding\_dim", 128, "Dimensionality of character embedding (default: 128)")  
tf.flags.DEFINE\_string("filter\_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')")  
tf.flags.DEFINE\_integer("num\_filters", 128, "Number of filters per filter size (default: 128)")  
tf.flags.DEFINE\_float("dropout\_keep\_prob", 0.5, "Dropout keep probability (default: 0.5)")  
tf.flags.DEFINE\_float("l2\_reg\_lambda", 0.0, "L2 regularization lambda (default: 0.0)")

# Training parameters  
tf.flags.DEFINE\_integer("batch\_size", 64, "Batch Size (default: 64)")  
tf.flags.DEFINE\_integer("num\_epochs", 200, "Number of training epochs (default: 200)")  
tf.flags.DEFINE\_integer("evaluate\_every", 100, "Evaluate model on dev set after this many steps (default: 100)")  
tf.flags.DEFINE\_integer("checkpoint\_every", 100, "Save model after this many steps (default: 100)")  
tf.flags.DEFINE\_integer("num\_checkpoints", 5, "Number of checkpoints to store (default: 5)")  
# Misc Parameters  
tf.flags.DEFINE\_boolean("allow\_soft\_placement", True, "Allow device soft device placement")  
tf.flags.DEFINE\_boolean("log\_device\_placement", False, "Log placement of ops on devices")

FLAGS = tf.flags.FLAGS  
FLAGS.\_parse\_flags()  
print("\nParameters:")  
for attr, value in sorted(FLAGS.\_\_flags.items()):  
    print("{}={}".format(attr.upper(), value))  
print("")

# Data Preparation  
# ==================================================

# Load data  
print("Loading data...")  
x\_text, y = data\_helpers.load\_data\_and\_labels()  
print(x\_text)  
# Build vocabulary  
max\_document\_length = max([len(x) for x in x\_text])  
vocab\_processor = learn.preprocessing.VocabularyProcessor(max\_document\_length)  
text\_list=[]  
for text in x\_text:  
    text\_list.append(' '.join(text))  
x = np.array(list(vocab\_processor.fit\_transform(text\_list)))  
print(x)  
# Randomly shuffle data  
np.random.seed(10)  
shuffle\_indices = np.random.permutation(np.arange(len(y)))  
x\_shuffled = x[shuffle\_indices]  
y\_shuffled = y[shuffle\_indices]

# Split train/test set  
# TODO: This is very crude, should use cross-validation  
dev\_sample\_index = -1 \* int(FLAGS.dev\_sample\_percentage \* float(len(y)))  
x\_train, x\_dev = x\_shuffled[:dev\_sample\_index], x\_shuffled[dev\_sample\_index:]  
y\_train, y\_dev = y\_shuffled[:dev\_sample\_index], y\_shuffled[dev\_sample\_index:]  
print("Vocabulary Size: {:d}".format(len(vocab\_processor.vocabulary\_)))  
print("Train/Dev split: {:d}/{:d}".format(len(y\_train), len(y\_dev)))

# Training  
# ==================================================

with tf.Graph().as\_default():  
    session\_conf = tf.ConfigProto(  
      allow\_soft\_placement=FLAGS.allow\_soft\_placement,  
      log\_device\_placement=FLAGS.log\_device\_placement)  
    sess = tf.Session(config=session\_conf)  
    with sess.as\_default():  
        cnn = TextCNN(  
            sequence\_length=x\_train.shape[1],  
            num\_classes=y\_train.shape[1],  
            vocab\_size=len(vocab\_processor.vocabulary\_),  
            embedding\_size=FLAGS.embedding\_dim,  
            filter\_sizes=list(map(int, FLAGS.filter\_sizes.split(","))),  
            num\_filters=FLAGS.num\_filters,  
            l2\_reg\_lambda=FLAGS.l2\_reg\_lambda)

        # Define Training procedure  
        global\_step = tf.Variable(0, name="global\_step", trainable=False)  
        optimizer = tf.train.AdamOptimizer(1e-3)  
        grads\_and\_vars = optimizer.compute\_gradients(cnn.loss)  
        train\_op = optimizer.apply\_gradients(grads\_and\_vars, global\_step=global\_step)

        # Keep track of gradient values and sparsity (optional)  
        grad\_summaries = []  
        for g, v in grads\_and\_vars:  
            if g is not None:  
                grad\_hist\_summary = tf.summary.histogram("{}/grad/hist".format(v.name), g)  
                sparsity\_summary = tf.summary.scalar("{}/grad/sparsity".format(v.name), tf.nn.zero\_fraction(g))  
                grad\_summaries.append(grad\_hist\_summary)  
                grad\_summaries.append(sparsity\_summary)  
        grad\_summaries\_merged = tf.summary.merge(grad\_summaries)

        # Output directory for models and summaries  
        timestamp = str(int(time.time()))  
        out\_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))  
        print("Writing to {}\n".format(out\_dir))

        # Summaries for loss and accuracy  
        loss\_summary = tf.summary.scalar("loss", cnn.loss)  
        acc\_summary = tf.summary.scalar("accuracy", cnn.accuracy)

        # Train Summaries  
        train\_summary\_op = tf.summary.merge([loss\_summary, acc\_summary, grad\_summaries\_merged])  
        train\_summary\_dir = os.path.join(out\_dir, "summaries", "train")  
        train\_summary\_writer = tf.summary.FileWriter(train\_summary\_dir, sess.graph)

        # Dev summaries  
        dev\_summary\_op = tf.summary.merge([loss\_summary, acc\_summary])  
        dev\_summary\_dir = os.path.join(out\_dir, "summaries", "dev")  
        dev\_summary\_writer = tf.summary.FileWriter(dev\_summary\_dir, sess.graph)

        # Checkpoint directory. Tensorflow assumes this directory already exists so we need to create it  
        checkpoint\_dir = os.path.abspath(os.path.join(out\_dir, "checkpoints"))  
        checkpoint\_prefix = os.path.join(checkpoint\_dir, "model")  
        if not os.path.exists(checkpoint\_dir):  
            os.makedirs(checkpoint\_dir)  
        saver = tf.train.Saver(tf.global\_variables(), max\_to\_keep=FLAGS.num\_checkpoints)

        # Write vocabulary  
        vocab\_processor.save(os.path.join(out\_dir, "vocab"))

        # Initialize all variables  
        sess.run(tf.global\_variables\_initializer())

        def train\_step(x\_batch, y\_batch):  
            """  
            A single training step  
            """  
            feed\_dict = {  
              cnn.input\_x: x\_batch,  
              cnn.input\_y: y\_batch,  
              cnn.dropout\_keep\_prob: FLAGS.dropout\_keep\_prob  
            }  
            \_, step, summaries, loss, accuracy = sess.run(  
                [train\_op, global\_step, train\_summary\_op, cnn.loss, cnn.accuracy],  
                feed\_dict)  
            time\_str = datetime.datetime.now().isoformat()  
            print("{}: step {}, loss {:g}, acc {:g}".format(time\_str, step, loss, accuracy))  
            train\_summary\_writer.add\_summary(summaries, step)

        def dev\_step(x\_batch, y\_batch, writer=None):  
            """  
            Evaluates model on a dev set  
            """  
            feed\_dict = {  
              cnn.input\_x: x\_batch,  
              cnn.input\_y: y\_batch,  
              cnn.dropout\_keep\_prob: 1.0  
            }  
            step, summaries, loss, accuracy = sess.run(  
                [global\_step, dev\_summary\_op, cnn.loss, cnn.accuracy],  
                feed\_dict)  
            time\_str = datetime.datetime.now().isoformat()  
            print("{}: step {}, loss {:g}, acc {:g}".format(time\_str, step, loss, accuracy))  
            if writer:  
                writer.add\_summary(summaries, step)

        # Generate batches  
        batches = data\_helpers.batch\_iter(  
            list(zip(x\_train, y\_train)), FLAGS.batch\_size, FLAGS.num\_epochs)  
        # Training loop. For each batch...  
        for batch in batches:  
            x\_batch, y\_batch = zip(\*batch)  
            train\_step(x\_batch, y\_batch)  
            current\_step = tf.train.global\_step(sess, global\_step)  
            if current\_step % FLAGS.evaluate\_every == 0:  
                print("\nEvaluation:")  
                dev\_step(x\_dev, y\_dev, writer=dev\_summary\_writer)  
                print("")  
            if current\_step % FLAGS.checkpoint\_every == 0:  
                path = saver.save(sess, checkpoint\_prefix, global\_step=current\_step)  
                print("Saved model checkpoint to {}\n".format(path))

网址获取改进

Jieba 字符类型转换

import urllib.request

response = urllib.request.urlopen("https://www.zhihu.com/question/67946658")

b = response.read()

b = str(b, encoding='utf-8')

import re

temp = b

#temp = temp.decode("utf8")

#string = re.sub("[\s+\.\!\/\_,$%^\*(+\"\']+|[+——！，。？、~@#￥%……&\*（）]+".decode("utf8"), "".decode("utf8"),temp)

string = re.sub("[A-Za-z0-9\[\`\~\!\@\#\$\^\&\\*\(\)\=\|\{\}\'\:\;\'\,\[\]\.\<\>\/\?\~\！\@\#\\\&\\*\%\-\"\+\\_\ ]", "", temp)

print (string)

'''

import urllib.rerquest

from bs4 import BeautifulSoup

url ='http://www.baidu.com/s'

values ={'wd':'wang'}

encoded\_param = urllib.request.urlencode(values)

full\_url = url +'?'+ encoded\_param

response = urllib.request.urlopen(full\_url)

soup =BeautifulSoup(response)

soup.find\_all('a')

'''

#!/usr/bin/python

# ^\_^ coding:utf8 ^\_^

import re

import requests

import traceback

from urllib.parse import quote

import sys, getopt

import importlib

importlib.reload(sys)

class crawler:

'''爬百度搜索结果的爬虫'''

url = u''

urls = []

o\_urls = []

html = ''

total\_pages = 5

current\_page = 0

next\_page\_url = ''

timeout = 60 #默认超时时间为60秒

headersParameters = { #发送HTTP请求时的HEAD信息，用于伪装为浏览器

'Connection': 'Keep-Alive',

'Accept': 'text/html, application/xhtml+xml, \*/\*',

'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',

'Accept-Encoding': 'gzip, deflate',

'User-Agent': 'Mozilla/6.1 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko'

}

def \_\_init\_\_(self, keyword):

self.url = u'https://www.baidu.com/baidu?wd='+quote(keyword)+'&tn=monline\_dg&ie=utf-8'

def set\_timeout(self, time):

'''设置超时时间，单位：秒'''

try:

self.timeout = int(time)

except:

pass

def set\_total\_pages(self, num):

'''设置总共要爬取的页数'''

try:

self.total\_pages = int(num)

except:

pass

def set\_current\_url(self, url):

'''设置当前url'''

self.url = url

def switch\_url(self):

'''切换当前url为下一页的url

若下一页为空，则退出程序'''

if self.next\_page\_url == '':

sys.exit()

else:

self.set\_current\_url(self.next\_page\_url)

def is\_finish(self):

'''判断是否爬取完毕'''

if self.current\_page >= self.total\_pages:

return True

else:

return False

def get\_html(self):

'''爬取当前url所指页面的内容，保存到html中'''

r = requests.get(self.url ,timeout=self.timeout, headers=self.headersParameters)

if r.status\_code==200:

self.html = r.text

self.current\_page += 1

else:

self.html = u''

print ('[ERROR]',self.url,u'get此url返回的http状态码不是200')

def get\_urls(self):

'''从当前html中解析出搜索结果的url，保存到o\_urls'''

o\_urls = re.findall('href\=\"(http\:\/\/www\.baidu\.com\/link\?url\=.\*?)\" class\=\"c\-showurl\"', self.html)

o\_urls = list(set(o\_urls)) #去重

self.o\_urls = o\_urls

#取下一页地址

next = re.findall(' href\=\"(\/s\?wd\=[\w\d\%\&\=\\_\-]\*?)\" class\=\"n\"', self.html)

if len(next) > 0:

self.next\_page\_url = 'https://www.baidu.com'+next[-1]

else:

self.next\_page\_url = ''

def get\_real(self, o\_url):

'''获取重定向url指向的网址'''

r = requests.get(o\_url, allow\_redirects = False) #禁止自动跳转

if r.status\_code == 302:

try:

return r.headers['location'] #返回指向的地址

except:

pass

return o\_url #返回源地址

def transformation(self):

'''读取当前o\_urls中的链接重定向的网址，并保存到urls中'''

self.urls = []

for o\_url in self.o\_urls:

self.urls.append(self.get\_real(o\_url))

def print\_urls(self):

'''输出当前urls中的url'''

for url in self.urls:

print (url)

def print\_o\_urls(self):

'''输出当前o\_urls中的url'''

for url in self.o\_urls:

print (url)

def run(self):

while(not self.is\_finish()):

c.get\_html()

c.get\_urls()

c.transformation()

c.print\_urls()

c.switch\_url()

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

help = 'baidu\_crawler.py -k <keyword> [-t <timeout> -p <total pages>]'

keyword = "江歌案"

timeout = None

totalpages = 10

try:

opts, args = getopt.getopt(sys.argv[1:], "hk:t:p:")

except getopt.GetoptError:

print(help)

sys.exit(2)

for opt, arg in opts:

if opt == '-h':

print(help)

sys.exit()

elif opt in ("-k", "--keyword"):

keyword = arg

elif opt in ("-t", "--timeout"):

timeout = arg

elif opt in ("-p", "--totalpages"):

totalpages = arg

if keyword == None:

print(help)

sys.exit()

c = crawler(keyword)

if timeout != None:

c.set\_timeout(timeout)

if totalpages != None:

c.set\_total\_pages(totalpages)

c.run()

#!/usr/bin/python

# ^\_^ coding:utf8 ^\_^

import re

import requests

import traceback

from urllib.parse import quote

import sys, getopt

import importlib

importlib.reload(sys)

t=0

doc=open('out.txt','w')

print ("",file=doc)

class crawler:

'''爬百度搜索结果的爬虫'''

url = u''

urls = []

o\_urls = []

html = ''

total\_pages = 5

current\_page = 0

next\_page\_url = ''

timeout = 60 #默认超时时间为60秒

headersParameters = { #发送HTTP请求时的HEAD信息，用于伪装为浏览器

'Connection': 'Keep-Alive',

'Accept': 'text/html, application/xhtml+xml, \*/\*',

'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',

'Accept-Encoding': 'gzip, deflate',

'User-Agent': 'Mozilla/6.1 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko'

}

def \_\_init\_\_(self, keyword):

self.url = u'https://www.baidu.com/baidu?wd='+quote(keyword)+'&tn=monline\_dg&ie=utf-8'

def set\_timeout(self, time):

'''设置超时时间，单位：秒'''

try:

self.timeout = int(time)

except:

pass

def set\_total\_pages(self, num):

'''设置总共要爬取的页数'''

try:

self.total\_pages = int(num)

except:

pass

def set\_current\_url(self, url):

'''设置当前url'''

self.url = url

def switch\_url(self):

'''切换当前url为下一页的url

若下一页为空，则退出程序'''

if self.next\_page\_url == '':

sys.exit()

else:

self.set\_current\_url(self.next\_page\_url)

def is\_finish(self):

'''判断是否爬取完毕'''

if self.current\_page >= self.total\_pages:

return True

else:

return False

def get\_html(self):

'''爬取当前url所指页面的内容，保存到html中'''

r = requests.get(self.url ,timeout=self.timeout, headers=self.headersParameters)

if r.status\_code==200:

self.html = r.text

self.current\_page += 1

else:

self.html = u''

print ('[ERROR]',self.url,u'get此url返回的http状态码不是200')

def get\_urls(self):

'''从当前html中解析出搜索结果的url，保存到o\_urls'''

o\_urls = re.findall('href\=\"(http\:\/\/www\.baidu\.com\/link\?url\=.\*?)\" class\=\"c\-showurl\"', self.html)

o\_urls = list(set(o\_urls)) #去重

self.o\_urls = o\_urls

#取下一页地址

next = re.findall(' href\=\"(\/s\?wd\=[\w\d\%\&\=\\_\-]\*?)\" class\=\"n\"', self.html)

if len(next) > 0:

self.next\_page\_url = 'https://www.baidu.com'+next[-1]

else:

self.next\_page\_url = ''

def get\_real(self, o\_url):

'''获取重定向url指向的网址'''

r = requests.get(o\_url, allow\_redirects = False) #禁止自动跳转

if r.status\_code == 302:

try:

return r.headers['location'] #返回指向的地址

except:

pass

return o\_url #返回源地址

def transformation(self):

'''读取当前o\_urls中的链接重定向的网址，并保存到urls中'''

self.urls = []

for o\_url in self.o\_urls:

self.urls.append(self.get\_real(o\_url))

def print\_urls(self):

'''输出当前urls中的url'''

for url in self.urls:

print (url)

doc=open('out.txt','a')

print (url,file=doc)

def print\_o\_urls(self):

'''输出当前o\_urls中的url'''

for url in self.o\_urls:

print (url)

doc=open('out.txt','a')

print (url,file=doc)

def run(self):

while(not self.is\_finish()):

c.get\_html()

c.get\_urls()

c.transformation()

c.print\_urls()

c.switch\_url()

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

help = 'baidu\_crawler.py -k <keyword> [-t <timeout> -p <total pages>]'

keyword = "江歌案"

timeout = None

totalpages = 10

try:

opts, args = getopt.getopt(sys.argv[1:], "hk:t:p:")

except getopt.GetoptError:

print(help)

sys.exit(2)

for opt, arg in opts:

if opt == '-h':

print(help)

sys.exit()

elif opt in ("-k", "--keyword"):

keyword = arg

elif opt in ("-t", "--timeout"):

timeout = arg

elif opt in ("-p", "--totalpages"):

totalpages = arg

if keyword == None:

print(help)

sys.exit()

c = crawler(keyword)

if timeout != None:

c.set\_timeout(timeout)

if totalpages != None:

c.set\_total\_pages(totalpages)

c.run()

网址获取中记事本存储与读取

存储 读取 记事本

#!/usr/bin/python

# ^\_^ coding:utf8 ^\_^

import re

import requests

import traceback

from urllib.parse import quote

import sys, getopt

import importlib

importlib.reload(sys)

t=0

doc=open('out.txt','w')

print ("",file=doc)

class crawler:

'''爬百度搜索结果的爬虫'''

url = u''

urls = []

o\_urls = []

html = ''

total\_pages = 5

current\_page = 0

next\_page\_url = ''

timeout = 60 #默认超时时间为60秒

headersParameters = { #发送HTTP请求时的HEAD信息，用于伪装为浏览器

'Connection': 'Keep-Alive',

'Accept': 'text/html, application/xhtml+xml, \*/\*',

'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',

'Accept-Encoding': 'gzip, deflate',

'User-Agent': 'Mozilla/6.1 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko'

}

def \_\_init\_\_(self, keyword):

self.url = u'https://www.baidu.com/baidu?wd='+quote(keyword)+'&tn=monline\_dg&ie=utf-8'

def set\_timeout(self, time):

'''设置超时时间，单位：秒'''

try:

self.timeout = int(time)

except:

pass

def set\_total\_pages(self, num):

'''设置总共要爬取的页数'''

try:

self.total\_pages = int(num)

except:

pass

def set\_current\_url(self, url):

'''设置当前url'''

self.url = url

def switch\_url(self):

'''切换当前url为下一页的url

若下一页为空，则退出程序'''

if self.next\_page\_url == '':

sys.exit()

else:

self.set\_current\_url(self.next\_page\_url)

def is\_finish(self):

'''判断是否爬取完毕'''

if self.current\_page >= self.total\_pages:

return True

else:

return False

def get\_html(self):

'''爬取当前url所指页面的内容，保存到html中'''

r = requests.get(self.url ,timeout=self.timeout, headers=self.headersParameters)

if r.status\_code==200:

self.html = r.text

self.current\_page += 1

else:

self.html = u''

print ('[ERROR]',self.url,u'get此url返回的http状态码不是200')

def get\_urls(self):

'''从当前html中解析出搜索结果的url，保存到o\_urls'''

o\_urls = re.findall('href\=\"(http\:\/\/www\.baidu\.com\/link\?url\=.\*?)\" class\=\"c\-showurl\"', self.html)

o\_urls = list(set(o\_urls)) #去重

self.o\_urls = o\_urls

#取下一页地址

next = re.findall(' href\=\"(\/s\?wd\=[\w\d\%\&\=\\_\-]\*?)\" class\=\"n\"', self.html)

if len(next) > 0:

self.next\_page\_url = 'https://www.baidu.com'+next[-1]

else:

self.next\_page\_url = ''

def get\_real(self, o\_url):

'''获取重定向url指向的网址'''

r = requests.get(o\_url, allow\_redirects = False) #禁止自动跳转

if r.status\_code == 302:

try:

return r.headers['location'] #返回指向的地址

except:

pass

return o\_url #返回源地址

def transformation(self):

'''读取当前o\_urls中的链接重定向的网址，并保存到urls中'''

self.urls = []

for o\_url in self.o\_urls:

self.urls.append(self.get\_real(o\_url))

def print\_urls(self):

'''输出当前urls中的url'''

for url in self.urls:

print (url)

doc=open('out.txt','a')

print (url,file=doc)

def print\_o\_urls(self):

'''输出当前o\_urls中的url'''

for url in self.o\_urls:

print (url)

doc=open('out.txt','a')

print (url,file=doc)

def run(self):

while(not self.is\_finish()):

c.get\_html()

c.get\_urls()

c.transformation()

c.print\_urls()

c.switch\_url()

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

help = 'baidu\_crawler.py -k <keyword> [-t <timeout> -p <total pages>]'

keyword = "江歌案"

timeout = None

totalpages = 10

try:

opts, args = getopt.getopt(sys.argv[1:], "hk:t:p:")

except getopt.GetoptError:

print(help)

sys.exit(2)

for opt, arg in opts:

if opt == '-h':

print(help)

sys.exit()

elif opt in ("-k", "--keyword"):

keyword = arg

elif opt in ("-t", "--timeout"):

timeout = arg

elif opt in ("-p", "--totalpages"):

totalpages = arg

if keyword == None:

print(help)

sys.exit()

c = crawler(keyword)

if timeout != None:

c.set\_timeout(timeout)

if totalpages != None:

c.set\_total\_pages(totalpages)

c.run()

import urllib.request

import re

import linecache

import os

count = len(open('out.txt','rU').readlines())

print (count)

os.mkdir("out")

for i in range (1,count+2):

content = linecache.getline('out.txt',i)

print(content)

filename="out\\out"+str(i)+".txt"

doc=open(filename,'a')

print (content,file=doc)

response = urllib.request.urlopen(content)

b = response.read()

b = str(b, encoding='utf-8')

temp = b

string = re.sub("[A-Za-z0-9\[\`\~\!\@\#\$\^\&\\*\(\)\=\|\{\}\'\:\;\'\,\[\]\.\<\>\/\?\~\！\@\#\\\&\\*\%\-\"\+\\_\\n\ \ \ ]", "", temp)

string=str(string)

#print (string)

doc=open(filename,'a',encoding='utf-8')

#print(string)

print (string,file=doc)

爬取成果展示

爬虫 百度

#!/usr/bin/python  
# ^\_^ coding:utf8 ^\_^

import re  
import requests  
import traceback  
from urllib.parse import quote  
import sys, getopt  
import importlib  
importlib.reload(sys)

t=0  
doc=open('out.txt','w')  
print ("",file=doc)

class crawler:  
    '''爬百度搜索结果的爬虫'''  
    url = u''  
    urls = []  
    o\_urls = []  
    html = ''  
    total\_pages = 5  
    current\_page = 0  
    next\_page\_url = ''  
    timeout = 60                    #默认超时时间为60秒  
    headersParameters = {    #发送HTTP请求时的HEAD信息，用于伪装为浏览器  
        'Connection': 'Keep-Alive',  
        'Accept': 'text/html, application/xhtml+xml, \*/\*',  
        'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',  
        'Accept-Encoding': 'gzip, deflate',  
        'User-Agent': 'Mozilla/6.1 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko'  
    }

    def \_\_init\_\_(self, keyword):  
        self.url = u'https://www.baidu.com/baidu?wd='+quote(keyword)+'&tn=monline\_dg&ie=utf-8'

    def set\_timeout(self, time):  
        '''设置超时时间，单位：秒'''  
        try:  
            self.timeout = int(time)  
        except:  
            pass

    def set\_total\_pages(self, num):  
        '''设置总共要爬取的页数'''  
        try:  
            self.total\_pages = int(num)  
        except:  
            pass

    def set\_current\_url(self, url):  
        '''设置当前url'''  
        self.url = url

    def switch\_url(self):  
        '''切换当前url为下一页的url  
           若下一页为空，则退出程序'''  
        if self.next\_page\_url == '':  
            sys.exit()  
        else:  
            self.set\_current\_url(self.next\_page\_url)

    def is\_finish(self):  
        '''判断是否爬取完毕'''  
        if self.current\_page >= self.total\_pages:  
            return True  
        else:  
            return False

    def get\_html(self):  
        '''爬取当前url所指页面的内容，保存到html中'''  
        r = requests.get(self.url ,timeout=self.timeout, headers=self.headersParameters)  
        if r.status\_code==200:  
            self.html = r.text  
            self.current\_page += 1  
        else:  
            self.html = u''  
            print ('[ERROR]',self.url,u'get此url返回的http状态码不是200')

    def get\_urls(self):  
        '''从当前html中解析出搜索结果的url，保存到o\_urls'''  
        o\_urls = re.findall('href\=\"(http\:\/\/www\.baidu\.com\/link\?url\=.\*?)\" class\=\"c\-showurl\"', self.html)  
        o\_urls = list(set(o\_urls))  #去重  
        self.o\_urls = o\_urls  
        #取下一页地址  
        next = re.findall(' href\=\"(\/s\?wd\=[\w\d\%\&\=\\_\-]\*?)\" class\=\"n\"', self.html)  
        if len(next) > 0:  
            self.next\_page\_url = 'https://www.baidu.com'+next[-1]  
        else:  
            self.next\_page\_url = ''

    def get\_real(self, o\_url):  
        '''获取重定向url指向的网址'''  
        r = requests.get(o\_url, allow\_redirects = False)    #禁止自动跳转  
        if r.status\_code == 302:  
            try:  
                return r.headers['location']    #返回指向的地址  
            except:  
                pass  
        return o\_url    #返回源地址

    def transformation(self):  
        '''读取当前o\_urls中的链接重定向的网址，并保存到urls中'''  
        self.urls = []  
        for o\_url in self.o\_urls:  
            self.urls.append(self.get\_real(o\_url))

    def print\_urls(self):  
        '''输出当前urls中的url'''  
        for url in self.urls:  
            print (url)  
            doc=open('out.txt','a')  
            print (url,file=doc)  
              
    def print\_o\_urls(self):  
        '''输出当前o\_urls中的url'''  
        for url in self.o\_urls:  
            print (url)  
            doc=open('out.txt','a')  
            print (url,file=doc)

    def run(self):  
        while(not self.is\_finish()):  
            c.get\_html()  
            c.get\_urls()  
            c.transformation()  
            c.print\_urls()  
            c.switch\_url()

if \_\_name\_\_ == '\_\_main\_\_':  
    help = 'baidu\_crawler.py -k <keyword> [-t <timeout> -p <total pages>]'  
    keyword = "江歌案"  
    timeout  = None  
    totalpages = 10  
    try:  
        opts, args = getopt.getopt(sys.argv[1:], "hk:t:p:")  
    except getopt.GetoptError:  
        print(help)  
        sys.exit(2)  
    for opt, arg in opts:  
        if opt == '-h':  
            print(help)  
            sys.exit()  
        elif opt in ("-k", "--keyword"):  
            keyword = arg  
        elif opt in ("-t", "--timeout"):  
            timeout = arg  
        elif opt in ("-p", "--totalpages"):  
            totalpages = arg  
    if keyword == None:  
        print(help)  
        sys.exit()

    c = crawler(keyword)  
    if timeout != None:  
        c.set\_timeout(timeout)  
    if totalpages != None:  
        c.set\_total\_pages(totalpages)  
    c.run()

数据判别尝试

全判别 Tensorflow

import tensorflow as tf

import numpy as np

class TextCNN(object):

"""

A CNN for text classification.

Uses an embedding layer, followed by a convolutional, max-pooling and softmax layer.

"""

def \_\_init\_\_(

self, sequence\_length, num\_classes, vocab\_size,

embedding\_size, filter\_sizes, num\_filters, l2\_reg\_lambda=0.0):

# Placeholders for input, output and dropout

self.input\_x = tf.placeholder(tf.int32, [None, sequence\_length], name="input\_x")

self.input\_y = tf.placeholder(tf.float32, [None, num\_classes], name="input\_y")

self.dropout\_keep\_prob = tf.placeholder(tf.float32, name="dropout\_keep\_prob")

# Keeping track of l2 regularization loss (optional)

l2\_loss = tf.constant(0.0)

# Embedding layer

with tf.device('/cpu:0'), tf.name\_scope("embedding"):

self.W = tf.Variable(

tf.random\_uniform([vocab\_size, embedding\_size], -1.0, 1.0),

name="W")

self.embedded\_chars = tf.nn.embedding\_lookup(self.W, self.input\_x)

self.embedded\_chars\_expanded = tf.expand\_dims(self.embedded\_chars, -1)

# Create a convolution + maxpool layer for each filter size

pooled\_outputs = []

for i, filter\_size in enumerate(filter\_sizes):

with tf.name\_scope("conv-maxpool-%s" % filter\_size):

# Convolution Layer

filter\_shape = [filter\_size, embedding\_size, 1, num\_filters]

W = tf.Variable(tf.truncated\_normal(filter\_shape, stddev=0.1), name="W")

b = tf.Variable(tf.constant(0.1, shape=[num\_filters]), name="b")

conv = tf.nn.conv2d(

self.embedded\_chars\_expanded,

W,

strides=[1, 1, 1, 1],

padding="VALID",

name="conv")

# Apply nonlinearity

h = tf.nn.relu(tf.nn.bias\_add(conv, b), name="relu")

# Maxpooling over the outputs

pooled = tf.nn.max\_pool(

h,

ksize=[1, sequence\_length - filter\_size + 1, 1, 1],

strides=[1, 1, 1, 1],

padding='VALID',

name="pool")

pooled\_outputs.append(pooled)

# Combine all the pooled features

num\_filters\_total = num\_filters \* len(filter\_sizes)

self.h\_pool = tf.concat(pooled\_outputs, 3)

self.h\_pool\_flat = tf.reshape(self.h\_pool, [-1, num\_filters\_total])

# Add dropout

with tf.name\_scope("dropout"):

self.h\_drop = tf.nn.dropout(self.h\_pool\_flat, self.dropout\_keep\_prob)

# Final (unnormalized) scores and predictions

with tf.name\_scope("output"):

W = tf.get\_variable(

"W",

shape=[num\_filters\_total, num\_classes],

initializer=tf.contrib.layers.xavier\_initializer())

b = tf.Variable(tf.constant(0.1, shape=[num\_classes]), name="b")

l2\_loss += tf.nn.l2\_loss(W)

l2\_loss += tf.nn.l2\_loss(b)

self.scores = tf.nn.xw\_plus\_b(self.h\_drop, W, b, name="scores")

self.predictions = tf.argmax(self.scores, 1, name="predictions")

# CalculateMean cross-entropy loss

with tf.name\_scope("loss"):

losses = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=self.scores, labels=self.input\_y)

self.loss = tf.reduce\_mean(losses) + l2\_reg\_lambda \* l2\_loss

# Accuracy

with tf.name\_scope("accuracy"):

correct\_predictions = tf.equal(self.predictions, tf.argmax(self.input\_y, 1))

self.accuracy = tf.reduce\_mean(tf.cast(correct\_predictions, "float"), name="accuracy")

基于RNN的数据判别方法

RNN

# coding: utf-8

from \_\_future\_\_ import print\_function

import os  
import sys  
import time  
from datetime import timedelta

import numpy as np  
import tensorflow as tf  
from sklearn import metrics

from rnn\_model import TRNNConfig, TextRNN  
from data.cnews\_loader import read\_vocab, read\_category, batch\_iter, process\_file, build\_vocab

base\_dir = 'data/cnews'  
train\_dir = os.path.join(base\_dir, 'cnews.train.txt')  
test\_dir = os.path.join(base\_dir, 'cnews.test.txt')  
val\_dir = os.path.join(base\_dir, 'cnews.val.txt')  
vocab\_dir = os.path.join(base\_dir, 'cnews.vocab.txt')

save\_dir = 'checkpoints/textrnn'  
save\_path = os.path.join(save\_dir, 'best\_validation')  # 最佳验证结果保存路径

def get\_time\_dif(start\_time):  
    """获取已使用时间"""  
    end\_time = time.time()  
    time\_dif = end\_time - start\_time  
    return timedelta(seconds=int(round(time\_dif)))

def feed\_data(x\_batch, y\_batch, keep\_prob):  
    feed\_dict = {  
        model.input\_x: x\_batch,  
        model.input\_y: y\_batch,  
        model.keep\_prob: keep\_prob  
    }  
    return feed\_dict

def evaluate(sess, x\_, y\_):  
    """评估在某一数据上的准确率和损失"""  
    data\_len = len(x\_)  
    batch\_eval = batch\_iter(x\_, y\_, 128)  
    total\_loss = 0.0  
    total\_acc = 0.0  
    for x\_batch, y\_batch in batch\_eval:  
        batch\_len = len(x\_batch)  
        feed\_dict = feed\_data(x\_batch, y\_batch, 1.0)  
        loss, acc = sess.run([model.loss, model.acc], feed\_dict=feed\_dict)  
        total\_loss += loss \* batch\_len  
        total\_acc += acc \* batch\_len

    return total\_loss / data\_len, total\_acc / data\_len

def train():  
    print("Configuring TensorBoard and Saver...")  
    # 配置 Tensorboard，重新训练时，请将tensorboard文件夹删除，不然图会覆盖  
    tensorboard\_dir = 'tensorboard/textrnn'  
    if not os.path.exists(tensorboard\_dir):  
        os.makedirs(tensorboard\_dir)

    tf.summary.scalar("loss", model.loss)  
    tf.summary.scalar("accuracy", model.acc)  
    merged\_summary = tf.summary.merge\_all()  
    writer = tf.summary.FileWriter(tensorboard\_dir)

    # 配置 Saver  
    saver = tf.train.Saver()  
    if not os.path.exists(save\_dir):  
        os.makedirs(save\_dir)

    print("Loading training and validation data...")  
    # 载入训练集与验证集  
    start\_time = time.time()  
    x\_train, y\_train = process\_file(train\_dir, word\_to\_id, cat\_to\_id, config.seq\_length)  
    x\_val, y\_val = process\_file(val\_dir, word\_to\_id, cat\_to\_id, config.seq\_length)  
    time\_dif = get\_time\_dif(start\_time)  
    print("Time usage:", time\_dif)

    # 创建session  
    session = tf.Session()  
    session.run(tf.global\_variables\_initializer())  
    writer.add\_graph(session.graph)

    print('Training and evaluating...')  
    start\_time = time.time()  
    total\_batch = 0  # 总批次  
    best\_acc\_val = 0.0  # 最佳验证集准确率  
    last\_improved = 0  # 记录上一次提升批次  
    require\_improvement = 1000  # 如果超过1000轮未提升，提前结束训练

    flag = False  
    for epoch in range(config.num\_epochs):  
        print('Epoch:', epoch + 1)  
        batch\_train = batch\_iter(x\_train, y\_train, config.batch\_size)  
        for x\_batch, y\_batch in batch\_train:  
            feed\_dict = feed\_data(x\_batch, y\_batch, config.dropout\_keep\_prob)

            if total\_batch % config.save\_per\_batch == 0:  
                # 每多少轮次将训练结果写入tensorboard scalar  
                s = session.run(merged\_summary, feed\_dict=feed\_dict)  
                writer.add\_summary(s, total\_batch)

            if total\_batch % config.print\_per\_batch == 0:  
                # 每多少轮次输出在训练集和验证集上的性能  
                feed\_dict[model.keep\_prob] = 1.0  
                loss\_train, acc\_train = session.run([model.loss, model.acc], feed\_dict=feed\_dict)  
                loss\_val, acc\_val = evaluate(session, x\_val, y\_val)  # todo

                if acc\_val > best\_acc\_val:  
                    # 保存最好结果  
                    best\_acc\_val = acc\_val  
                    last\_improved = total\_batch  
                    saver.save(sess=session, save\_path=save\_path)  
                    improved\_str = '\*'  
                else:  
                    improved\_str = ''

                time\_dif = get\_time\_dif(start\_time)  
                msg = 'Iter: {0:>6}, Train Loss: {1:>6.2}, Train Acc: {2:>7.2%},' \  
                      + ' Val Loss: {3:>6.2}, Val Acc: {4:>7.2%}, Time: {5} {6}'  
                print(msg.format(total\_batch, loss\_train, acc\_train, loss\_val, acc\_val, time\_dif, improved\_str))

            session.run(model.optim, feed\_dict=feed\_dict)  # 运行优化  
            total\_batch += 1

            if total\_batch - last\_improved > require\_improvement:  
                # 验证集正确率长期不提升，提前结束训练  
                print("No optimization for a long time, auto-stopping...")  
                flag = True  
                break  # 跳出循环  
        if flag:  # 同上  
            break

def test():  
    print("Loading test data...")  
    start\_time = time.time()  
    x\_test, y\_test = process\_file(test\_dir, word\_to\_id, cat\_to\_id, config.seq\_length)

    session = tf.Session()  
    session.run(tf.global\_variables\_initializer())  
    saver = tf.train.Saver()  
    saver.restore(sess=session, save\_path=save\_path)  # 读取保存的模型

    print('Testing...')  
    loss\_test, acc\_test = evaluate(session, x\_test, y\_test)  
    msg = 'Test Loss: {0:>6.2}, Test Acc: {1:>7.2%}'  
    print(msg.format(loss\_test, acc\_test))

    batch\_size = 128  
    data\_len = len(x\_test)  
    num\_batch = int((data\_len - 1) / batch\_size) + 1

    y\_test\_cls = np.argmax(y\_test, 1)  
    y\_pred\_cls = np.zeros(shape=len(x\_test), dtype=np.int32)  # 保存预测结果  
    for i in range(num\_batch):  # 逐批次处理  
        start\_id = i \* batch\_size  
        end\_id = min((i + 1) \* batch\_size, data\_len)  
        feed\_dict = {  
            model.input\_x: x\_test[start\_id:end\_id],  
            model.keep\_prob: 1.0  
        }  
        y\_pred\_cls[start\_id:end\_id] = session.run(model.y\_pred\_cls, feed\_dict=feed\_dict)

    # 评估  
    print("Precision, Recall and F1-Score...")  
    print(metrics.classification\_report(y\_test\_cls, y\_pred\_cls, target\_names=categories))

    # 混淆矩阵  
    print("Confusion Matrix...")  
    cm = metrics.confusion\_matrix(y\_test\_cls, y\_pred\_cls)  
    print(cm)

    time\_dif = get\_time\_dif(start\_time)  
    print("Time usage:", time\_dif)

if \_\_name\_\_ == '\_\_main\_\_':  
    if len(sys.argv) != 2 or sys.argv[1] not in ['train', 'test']:  
        raise ValueError("""usage: python run\_rnn.py [train / test]""")

    print('Configuring RNN model...')  
    config = TRNNConfig()  
    if not os.path.exists(vocab\_dir):  # 如果不存在词汇表，重建  
        build\_vocab(train\_dir, vocab\_dir, config.vocab\_size)  
    categories, cat\_to\_id = read\_category()  
    words, word\_to\_id = read\_vocab(vocab\_dir)  
    config.vocab\_size = len(words)  
    model = TextRNN(config)

    if sys.argv[1] == 'train':  
        train()  
    else:  
        test()

#!/usr/bin/python  
# -\*- coding: utf-8 -\*-

import tensorflow as tf

class TRNNConfig(object):  
    """RNN配置参数"""

    # 模型参数  
    embedding\_dim = 64      # 词向量维度  
    seq\_length = 600        # 序列长度  
    num\_classes = 10        # 类别数  
    vocab\_size = 5000       # 词汇表达小

    num\_layers= 2           # 隐藏层层数  
    hidden\_dim = 128        # 隐藏层神经元  
    rnn = 'gru'             # lstm 或 gru

    dropout\_keep\_prob = 0.8 # dropout保留比例  
    learning\_rate = 1e-3    # 学习率

    batch\_size = 128         # 每批训练大小  
    num\_epochs = 10          # 总迭代轮次

    print\_per\_batch = 100    # 每多少轮输出一次结果  
    save\_per\_batch = 10      # 每多少轮存入tensorboard

class TextRNN(object):  
    """文本分类，RNN模型"""  
    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.float32, [None, self.config.num\_classes], name='input\_y')  
        self.keep\_prob = tf.placeholder(tf.float32, name='keep\_prob')

        self.rnn()

    def rnn(self):  
        """rnn模型"""

        def lstm\_cell():   # lstm核  
            return tf.contrib.rnn.BasicLSTMCell(self.config.hidden\_dim, state\_is\_tuple=True)

        def gru\_cell():  # gru核  
            return tf.contrib.rnn.GRUCell(self.config.hidden\_dim)

        def dropout(): # 为每一个rnn核后面加一个dropout层  
            if (self.config.rnn == 'lstm'):  
                cell = lstm\_cell()  
            else:  
                cell = gru\_cell()  
            return tf.contrib.rnn.DropoutWrapper(cell, output\_keep\_prob=self.keep\_prob)

        # 词向量映射  
        with tf.device('/cpu:0'):  
            embedding = tf.get\_variable('embedding', [self.config.vocab\_size, self.config.embedding\_dim])  
            embedding\_inputs = tf.nn.embedding\_lookup(embedding, self.input\_x)

        with tf.name\_scope("rnn"):  
            # 多层rnn网络  
            cells = [dropout() for \_ in range(self.config.num\_layers)]  
            rnn\_cell = tf.contrib.rnn.MultiRNNCell(cells, state\_is\_tuple=True)

            \_outputs, \_ = tf.nn.dynamic\_rnn(cell=rnn\_cell, inputs=embedding\_inputs, dtype=tf.float32)  
            last = \_outputs[:, -1, :]  # 取最后一个时序输出作为结果

        with tf.name\_scope("score"):  
            # 全连接层，后面接dropout以及relu激活  
            fc = tf.layers.dense(last, self.config.hidden\_dim, name='fc1')  
            fc = tf.contrib.layers.dropout(fc, self.keep\_prob)  
            fc = tf.nn.relu(fc)

            # 分类器  
            self.logits = tf.layers.dense(fc, self.config.num\_classes, name='fc2')  
            self.y\_pred\_cls = tf.argmax(tf.nn.softmax(self.logits), 1)  # 预测类别

        with tf.name\_scope("optimize"):  
            # 损失函数，交叉熵  
            cross\_entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=self.logits, labels=self.input\_y)  
            self.loss = tf.reduce\_mean(cross\_entropy)  
            # 优化器  
            self.optim = tf.train.AdamOptimizer(learning\_rate=self.config.learning\_rate).minimize(self.loss)

        with tf.name\_scope("accuracy"):  
            # 准确率  
            correct\_pred = tf.equal(tf.argmax(self.input\_y, 1), self.y\_pred\_cls)  
            self.acc = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))